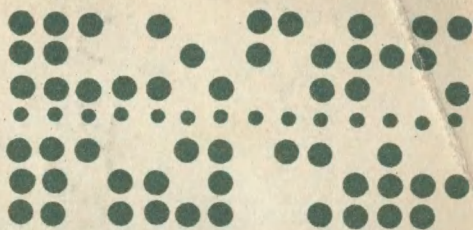


**БИБЛИОТЕЧКА
ПРОГРАММИСТА**



В. Я. КАРПОВ

Алгоритмический язык ФОРТРАН



БИБЛИОТЕЧКА ПРОГРАММИСТА

В. Я. КАРПОВ

АЛГОРИТМИЧЕСКИЙ ЯЗЫК ФОРТРАН

ФОРТРАН-ДУБНА

Под редакцией
Н. Н. ГОВОРУНА



ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
Москва 1976

Алгоритмический язык фортран (фортран-Дубна).
Карпов В. Я. Главная редакция физико-математической литературы изд-ва «Наука», М., 1976.

В книге дано подробное описание языка фортран, одного из самых распространенных языков программирования на электронных вычислительных машинах. Материал расположен по принципу перехода от простого к сложному и разбит на две части. Первая часть знакомит читателя с основами языка фортран. В ней объясняются понятия, общие для всех версий языка. Вторая часть содержит подробное описание дубненской версии фортрана и может служить развернутым справочным пособием для лиц, непосредственно занимающихся программированием. В приложении к основному материалу помещена сводка отличий дубненской версии языка фортран от фортрана IV, благодаря которой книга будет полезна также и для программистов, работающих на машинах серии ЕС.

Владимир Яковлевич Карпов

Алгоритмический язык фортран (фортран-Дубна)

(Серия «Библиотечка программиста»)

М., 1976 г., 192 стр. с илл.

Редактор Г. Я. Пирогова

Технический редактор И. Ш. Аксельрод

Корректор Е. Я. Строева

Сдано в набор 7/V 1976 г. Подписано к печати 31/VIII 1976 г.
Бумага типографская № 2 84×108¹/₃₂. Физ. печ. л. 6. Условн.
печ. л. 10,08. Уч.-изд. л. 9,48. Тираж 100 000 экз. Т-15758.
Заказ № 144. Цена книги 57 коп.

Издательство «Наука»

Главная редакция
физико-математической литературы
117071, Москва, В-71, Ленинский проспект, 15

4-я типография издательства «Наука»,
Новосибирск, 77, ул. Станиславского, д. 25.

К $\frac{20204-119}{053(02)-76}$ 71—76

© Главная редакция
физико-математической литературы
издательства «Наука», 1976

ОГЛАВЛЕНИЕ

Предисловие редактора	5
Введение	7
Часть I. Элементы языка фортран	11
§ 1. Алфавит языка фортран	11
§ 2. Константы	13
§ 3. Переменные	15
§ 4. Арифметические выражения	17
§ 5. Оператор присваивания	19
§ 6. Запись программы на бланках	21
§ 7. Ввод и вывод информации	28
§ 8. Оформление программы	37
§ 9. Операторы передачи управления	40
9.1. Оператор безусловной передачи управления (40). 9.2. Арифметический оператор условной передачи управления (40). 9.3. Оператор цикла (41).	
§ 10. Массивы переменных	45
§ 11. Функции и подпрограммы	51
11.1. Функция-оператор (52). 11.2. Функция-подпрограмма (54). 11.3. Подпрограмма (58). 11.4. Оператор COMMON (60).	
Часть II. Фортран-Дубна	64
§ 12. Хранение информации в машине БЭСМ-6	65
§ 13. Константы	71
13.1. Целые константы (71). 13.2. Вещественные константы (72). 13.3. Константы с двойной точностью (73). 13.4. Комплексные константы (74). 13.5. Восьмеричные константы (75). 13.6. Логические константы (75). 13.7. Холлеритовские (текстовые) константы (76).	
§ 14. Переменные	78
14.1. Типы переменных (78). 14.2. Массивы переменных (80). 14.3. Расположение массива в памяти машины (81). 14.4. Допустимые формы индексов (83). 14.5. Эквивалентность переменных (85). 14.6. Оператор DATA (88).	

§ 15. Выражения	91
15.1. Арифметические выражения (92)	
15.2. Выражения отношения (94).	
15.3. Логические выражения (94).	
§ 16. Операторы	96
16.1. Исполняемые и неисполняемые операторы (97).	
16.2. Операторы присваивания (98).	
16.3. Операторы передачи управления (100).	
16.4. Операторы цикла (103).	
§ 17. Операторы ввода и вывода информации	110
17.1. Операторы вывода (112).	
17.2. Операторы ввода (115).	
17.3. Список оператора ввода/вывода (116).	
17.4. Управление магнитной лентой (118).	
17.5. Операторы ENCODE и DECODE (120).	
§ 18. Спецификации формата	122
18.1. Спецификация <i>In</i> (123).	
18.2. Спецификация <i>Fn. m</i> (125).	
18.3. Спецификация <i>En. m</i> (128).	
18.4. Спецификация <i>Dn. m</i> (131).	
18.5. Спецификация <i>On</i> (132).	
18.6. Спецификация <i>Ln</i> (135).	
18.7. Спецификация <i>Al</i> (136).	
18.8. Масштабный коэффициент (139).	
18.9. Управляющие и редакционные спецификации (141).	
18.10. Повторяемые спецификации (145).	
18.11. Оператор FORMAT (145).	
18.12. Переменный список спецификаций оператора FORMAT (149).	
§ 19. Функции и подпрограммы	152
19.1. Основная программа и подпрограммы (152).	
19.2. Библиотечные функции (153).	
19.3. Функция-оператор (154).	
19.4. Функция-подпрограмма (156).	
19.5. Подпрограмма (160).	
19.6. Оператор EXTERNAL (164).	
19.7. Оператор ENTRY (165).	
19.8. Оператор COMMON (167).	
19.9. BLOCK DATA (170).	
19.10. Соответствие массивов программных единиц (171).	
Приложения	
1. Кодировка символов	174
2. Отличия фортрана-Дубна от фортрана IV	176
3. Библиотечные функции-подпрограммы	184
Ответы к упражнениям	188

ПРЕДИСЛОВИЕ РЕДАКТОРА

Язык фортран является одним из самых распространенных за рубежом языков программирования, а в последнее десятилетие он стал пользоваться все большей популярностью и в нашей стране. В настоящее время практически все ЭВМ, выпускаемые отечественной промышленностью, имеют в составе математического обеспечения трансляторы с языка фортран. Готовится утверждение Государственного стандарта на этот язык.

Рост популярности языка выражается в постоянном увеличении числа работающих на нем программистов. В связи с этим стала ощущаться острая нехватка учебных пособий. Особенно трудное положение с литературой по языку фортран сложилось для пользователей, работающих на БЭСМ-6. Изданное весьма малым тиражом описание языка фортран под редакцией В. П. Широкова стало библиографической редкостью.

Предлагаемая читателю книга восполняет пробел в этой литературе на русском языке. Книга по стилю написания носит характер учебного пособия и в значительной степени независима от реализации языка на конкретной ЭВМ. Автор достиг определенной популярности изложения материала, однако документально точно дал информацию о версии языка

фортран для ЭВМ БЭСМ-6. Таким образом, книга вполне может служить и в качестве справочника для программистов, работающих на БЭСМ-6.

Многочисленные примеры и упражнения, которыми иллюстрируются понятия и конструкции языка, помогут читателю овладеть практическими навыками программирования.

Книга, несомненно, будет полезна широкому кругу читателей, связанных с работой на вычислительных машинах, а также всем, кто интересуется вопросами программирования на ЭВМ.

Н. Н. Говорун

ВВЕДЕНИЕ

В настоящее время электронные вычислительные машины (ЭВМ) используются для решения самых различных задач науки, техники, экономики. Широкое распространение ЭВМ связано с их фантастическим быстродействием и способностью хранить огромные объемы информации.

Обычная вычислительная машина является устройством, производящим операции над числами согласно заданной ей программе действий. Количество операций, которые умеет выполнять машина, довольно ограничено. Машина может складывать числа, вычитать, делить, умножать, пересылать число из одного места памяти в другое и т. п. Кроме того, она умеет читать информацию с перфокарт, бумажных или магнитных лент, магнитных барабанов и дисков, выводить на них информацию, а также печатать результаты вычислений на бумаге.

Чтобы решить на машине конкретную задачу, ее нужно представить в виде последовательности инструкций-команд. Каждая команда указывает на выполнение одной операции, а их последовательность, называемая программой, определяет работу машины. Поскольку машина производит лишь элементарные действия, любая проблема может быть решена на машине только в том случае, если она сведена к совокупности таких элементарных действий. Имеется много способов сведения к арифметическим операциям задач интегрирования, дифференцирования, решения уравнений и т. д. Эти способы изучаются прикладной математикой и называются численными методами. После выбора подходящего метода задача

формулируется как перечень определенных действий (так называемого алгоритма), выполнение которых приводит к решению задачи. Разработав алгоритм, можно переходить, собственно, к программированию, т. е. к составлению списка приказов-команд, имеющих понятную для машины форму записи. Машина понимает только команды, записанные специальным образом в виде последовательности цифр. В этой последовательности кодируется операция (сложение, умножение и т. п.), которую должна выполнить машина, адреса чисел, над которыми нужно произвести операцию, и некоторая дополнительная информация. Такого рода процесс программирования называется программированием в машинных кодах. Программирование в кодах является очень кропотливой, утомительной и в то же время довольно механической работой.

Попытки облегчить процесс программирования привели к созданию алгоритмических языков, позволяющих записывать алгоритм решения задачи в форме, более удобной и привычной для программиста, чем язык машинных кодов. Программа, записанная на алгоритмическом языке, представляет собой последовательность предложений, или операторов, оформленных по определенным правилам. Такая программа не может быть непосредственно воспринята вычислительной машиной, которая понимает только язык кодов. Перевод программы с алгоритмического языка на язык машинных кодов производится автоматически специальной программой, называемой транслятором*) с данного языка. Таким образом, транслятор позволяет автоматизировать процесс кодировки программы. В общих чертах процесс трансляции состоит в следующем. Транслятор последовательно читает перфокарты, на которых пробиты предложения программы. Содержимое каждой перфокарты обычно выводится в виде одной строки на печатающее устройство для того, чтобы программист мог контролировать текст, пробитый на перфокартах. По прочтении одного предложения транслятор проверяет, нет ли в нем грамматических ошибок, т. е. соответствует ли конструкция предложения правилам языка. Если

*) От английского слова to translate — переводить.

ошибка обнаружена, то в строке, следующей за текстом предложения на выдаче, печатается сообщение об ошибке (так называемая диагностика транслятора) с указанием того, какого рода ошибка была обнаружена. Грамматически неправильные предложения машиной игнорируются. Каждое грамматически правильное предложение переводится с алгоритмического языка на язык машинных кодов, и получающаяся таким образом программа в кодах запоминается машиной. Процесс трансляции продолжается до тех пор, пока машина не встретит специальную карту, указывающую на то, что программа введена полностью. После этого процесс трансляции прекращается и машина может приступить к выполнению программы, записанной теперь в машинных кодах.

Одним из алгоритмических языков, получивших в настоящее время наибольшее распространение, является фортран (Mathematical Formula TRANslating System). Этот язык был разработан сотрудниками фирмы IBM (США) в 1954—1956 годах для машины IBM 704. В первом документе, посвященном фортрану и выпущенном в ноябре 1954 года, было сказано: «Система трансляции математических формул, сокращенно фортран, будет состоять из большого количества программ, позволяющих IBM 704 воспринимать сжатую формулировку задачи в терминах математических обозначений и составлять автоматически высокоэффективную программу для решения задачи».

В июне 1958 года появилось сообщение о создании новой версии языка со значительными добавлениями. Эта версия была названа фортраном II и содержала понятие подпрограммы, представленное операторами SUBROUTINE, FUNCTION, CALL, RETURN, END. Был введен оператор COMMON для обеспечения связи между программными единицами.

В 1962 году был выпущен предварительный бюллетень с описанием версии, которая теперь называется фортран IV. По сравнению с фортраном II в ней добавились понятие типа величин (операторы INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL), логический оператор условного перехода, оператор DATA и BLOCK DATA. Некоторые из этих понятий были определены в вариантах фортрана II,

но только в фортране IV они стали неотъемлемой частью языка. В то же время в фортран IV не вошли некоторые машинно-зависимые операторы и вследствие этого фортран II не стал подмножеством фортрана IV в том смысле, что программа, написанная на фортране II, может не удовлетворять правилам языка фортран IV.

В мае 1962 года начала работу комиссия при Американской Ассоциации Стандартов (ASA), в результате деятельности которой были созданы два стандарта, официально известные как фортран и основной фортран (basic fortran). Эти языки приблизительно соответствуют фортранам IV и II, однако с тем существенным отличием, что основной фортран является подмножеством фортрана.

Кроме перечисленных выше версий языка, имеется большое количество его вариантов для различных машин. В Советском Союзе широко эксплуатируется так называемая дубненская версия языка фортран, транслятор с которой установлен на машине БЭСМ-6. При создании этого транслятора в качестве прототипа была взята версия фортран-CERN машин серии CDC. Благодаря этому для машины БЭСМ-6 стала доступной обширная библиотека стандартных программ Европейского Центра Ядерных Исследований (CERN).

Дальнейшее изложение ориентировано именно на дубненскую версию языка, которую мы будем называть сокращенно фортран-Дубна.

ЧАСТЬ I

ЭЛЕМЕНТЫ ЯЗЫКА ФОРТРАН

Как уже говорилось во введении, в настоящее время существуют несколько версий языка фортран. Все они имеют одинаковые основные правила и отличаются друг от друга лишь присутствием дополнительных возможностей или некоторых ограничений. В первой части книги объясняются элементарные понятия языка, присутствующие во всех его версиях. В этой части мы не будем стремиться к строгости изложения, а постараемся на примерах и аналогиях дать читателю общее представление о фортране, с тем чтобы он смог легко разобраться в любой конкретной его версии.

Программистам, работающим на дубненской версии языка и желающим ознакомиться с ее синтаксисом, мы рекомендуем эту часть не читать, а сразу обратиться ко второй части книги.

§ 1. Алфавит языка фортран

При записи предложений любого языка используется набор некоторых допустимых символов — букв этого языка, называемый алфавитом. Так, русский алфавит содержит 33 буквы от А до Я, английский алфавит 26 букв от А до Z, а греческий алфавит составляют 24 буквы от α до ω . Язык фортран также имеет свой набор допустимых символов, которые могут быть использованы для написания программы. За основу алфавита языка приняты латинские буквы.

При записи предложений (операторов) фортрана допускаются:

а) 26 прописных букв латинского алфавита:

A, B, C, D, E, F, G, H, I,
J, K, L, M, N, O, P, Q, R,
S, T, U, V, W, X, Y, Z;

б) 10 арабских цифр:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9;

в) служебные знаки:

+ плюс,
— минус,
/ слэш (знак деления),
* звездочка (знак умножения),
(открывающая скобка,
) закрывающая скобка,
= знак присваивания,
. точка,
, запятая,
\$ знак доллара,
' апостроф.

В фортране не различают строчные и прописные буквы, и поэтому принято пользоваться только прописными буквами. Когда программу записывают на бланках от руки, то для того, чтобы отличать цифру ноль от буквы O, ноль перечеркивают и пишут Ø. Такое соглашение не является общепринятым: иногда перечеркивают букву O, но в каждом случае значение символов O и Ø легко понять по смыслу операторов.

Отметим, что некоторые версии языка имеют отличия в алфавите от описанного выше. В фортране IV знак доллара считается буквой и допускается в идентификаторах переменных (см. § 3 и приложение 3). В дубненской версии языка для удобства программирования наравне с буквами латинского алфавита разрешено использование прописных букв русского алфавита. Кроме того, в этой версии в комментариях и текстах допускаются все символы, имеющиеся в устройстве для подготовки перфокарт (УПП) и на автоматическом цифровом печатающем устройстве (АЦПУ).

Запись предложений программы на языке фортран ведется в строку. Не разрешается подчеркивать или надчеркивать символы. Нельзя обычным образом писать показатель степени и ставить нижние или верхние индексы. О способе записи операторов будет подробно рассказано в § 6.

§ 2. Константы

Электронная вычислительная машина хранит любую информацию в двоичном коде, как последовательность нулей и единиц. Местом хранения информации служат ячейки памяти машины. Однако при программировании на языке фортран можно ничего не знать ни о представлении чисел внутри машины, ни об устройстве ее памяти. В программе числа записываются в обычном десятичном виде с той лишь разницей, что вместо запятой, отделяющей целую часть от дробной, в фортране используется точка.

Пример.

Число	Запись на фортране
3	3
2,565	2.565
705,6	705.6

Числа хранятся машиной с определенным числом знаков, зависящим от длины ячейки ее памяти. Поэтому в программировании нет иррациональных чисел. Все числа рациональны и могут иметь обычно неограниченное количество десятичных цифр. Например, при делении единицы на три на машине БЭСМ-6, которая хранит двенадцать знаков числа, мы получим не бесконечную дробь $0,333...$, а число $0,333...33$, имеющее двенадцать знаков после запятой. Если это число умножить на три, то результатом будет число $0,999...99$ (двенадцать девяток), а не единица. Таким образом, при машинном делении, а затем умножении единицы на три получается не единица, а дробь, отличающаяся от единицы в последнем знаке числа.

Ошибки такого рода, возникающие в вычислениях, называются ошибками округления. Накапливаясь, они могут сильно исказить результаты расчетов, иногда

вообще привести к неверному ответу. Ошибки округления специально изучаются в численных методах. С программистской точки зрения важно, что наличие ошибок округления может повлиять не только на результат расчета, но и на сам процесс вычисления. Например, мы вычисляем в программе номер какого-либо элемента, а затем хотим написать условие: «если номер элемента равен пяти, то нужно делать то-то». Из-за ошибок округления мы не можем быть уверены, что номер элемента окажется целым и не будет содержать отличных от нуля цифр в младших десятичных разрядах. Чтобы условие выполнилось правильно, нам придется как-то объяснить машине, что номера 4,999...99 и 5,000...01 это все равно, что номер 5.

В языке фортран указанное неудобство исключается путем введения двух типов чисел: целых и вещественных. Числа разных типов по-разному хранятся машиной. Понятие вещественного числа в программировании несколько отличается от общеупотребительного и означает, что такое число имеет дробную часть, в отличие от числа типа «целое». Поэтому в записи вещественного числа обязательно присутствует десятичная точка. При этом, если дробная часть не указана, она считается равной нулю. Например, запись 1. означает то же самое, что и 1.0.

Числа, используемые в программе, принято называть константами, поскольку они являются величинами, не меняющимися в процессе счета. Если константа не имеет знака, она считается положительной.

Вещественные константы характеризуются при записи наличием десятичной точки. Например,

—10. 52.5 +.001

Знак плюс перед положительной константой можно опускать. Поскольку в фортране запись предложений ведется в строку и нельзя обычным образом ставить показатель степени, то для указания десятичного порядка вещественного числа используется буква E. Например, число $5,6 \cdot 10^3$ на фортране записывается как 5.6E+3 или 5.6E3. Число, указывающее порядок, должно быть целым.

Примеры. Правильная запись констант

Целые	Вещественные
125	5692.00
-1500	5.692E + 3
+7	569200.E - 2

Неправильная запись констант

Целые	Вещественные
125.	2
2E + 5	2,6
12.35	5.68E + 2.

Упражнения.

1. Записать следующие числа в виде вещественных констант фортрана:

8	$-1,16 \cdot 10^{-11}$	347,4	-0,000028
552,6	10^5	3,14159	

2. Какие из указанных ниже констант не являются целыми? Почему?

8	8.	+ 526	- 211	1.E + 5
---	----	-------	-------	---------

3. Какие из указанных ниже констант не являются вещественными? Почему?

5.638	-256	1.64153E - 11
-1.00	1.999E + 2	

4. Проверьте, имеют ли приведенные ниже пары вещественных констант одинаковое значение:

360.	$3.6E + 2$	1.0	1.
-0.027	-270.E - 4	905.796E0	905.796
+1.1363	1.1363E - 1	0.000001	1.E - 6

§ 3. Переменные

При составлении программы часто бывает удобным запомнить некоторое число в определенном месте в памяти машины, с тем чтобы его можно было использовать при дальнейших вычислениях. Очень важно различать такие понятия, как «определенное место в памяти машины», или «ячейка» и «содержимое ячейки». Ячейку памяти можно представлять себе как ящик, в котором в каждый момент времени хранится только одно число. В процессе проведения расчета содержимое конкретной ячейки может меняться, и поэтому о ней говорят как о переменной. В языке фортран переменным можно присваивать имена. Эти имена называются *идентификаторами*. Так, некоторую

переменную мы можем назвать SUMM и хранить в ней число 5,89 на одном этапе выполнения программы, число 6,92 на другом этапе и т. д.

Идентификатором может служить любая комбинация от одной до шести букв или цифр, причем первым символом этой комбинации должна быть буква. Использование символов, не являющихся буквами или цифрами, в идентификаторе не допускается.

Примеры. Правильная запись идентификаторов

I	MATRIX	ALFA
C	K 142	U12A

Неправильная запись идентификаторов

4U21 — идентификатор должен начинаться с буквы;
VELOCITY — в идентификаторе должно быть не более шести символов;
A12.5 — нельзя использовать символ, не являющийся буквой или цифрой.

Идентификатор переменной выполняет одновременно две различные функции:

- 1) определяет место переменной в памяти машины;
- 2) указывает на то, какого вида информация хранится в этом месте.

Как уже говорилось в § 2, число в машине может храниться как целое или как вещественное. Если не оговорено противное (см. часть II), то переменные, идентификаторы которых начинаются с букв I, J, K, L, M, N, имеют целые значения. Идентификаторы, не начинающиеся с одной из этих букв, считаются идентификаторами вещественных переменных.

Примеры. Идентификаторы целых переменных

I	INDEX	MATRIX
---	-------	--------

Идентификаторы вещественных переменных

ALFA	HEIGHT	AREA
------	--------	------

Упражнения. Укажите, какие из приведенных ниже комбинаций символов могут служить идентификаторами целых или вещественных переменных и какие нельзя использовать в качестве идентификаторов:

ARRAY	BETA2	IP2	EPSILON
I(2)	IFIP	BESM-6	DRUM
DELTA	AJAX	MARIA	421
HUME	A	C1.2	KAPPA
ALGOL	FORTRAN	COBOL	MADLEN

§ 4. Арифметические выражения

В фортране арифметические операции обозначаются следующим образом:

- + сложение,
- вычитание,
- * умножение,
- / деление,
- ** возведение в степень.

Поэтому,

$A+B$ обозначает сумму величин A и B ,

$A-B$ — разность величин A и B ,

$A*B$ — произведение величин A и B (знак $*$ не может быть опущен: AB есть идентификатор, а не произведение A и B),

A/B обозначает частное от деления A на B .

$A**B$ — число, получающееся при возведении A в степень B .

Отметим, что результатом действий над целыми величинами является также целая величина, получаемая, если это необходимо, отбрасыванием дробной части числа, а не округлением. Например, вычисление выражения $2/3$ (здесь 2 и 3 — целые константы) дает нуль (также целую величину).

Если нужно выполнить несколько арифметических операций, их можно записывать, как обычно, в строку, используя скобки для указания последовательности выполнения операций.

Примеры записи выражений.

Алгебраическая запись

Запись на фортране

$$\frac{a \cdot c}{b}$$
$$(x+a)^3$$
$$x+a^2$$
$$\frac{a}{b \cdot x}$$

$$A \cdot C / B$$

$$(X+A)**3$$

$$X+A**2$$

$$A/B/X \text{ или } A/(B \cdot X)$$

Если последовательность выполнения операций не определена скобками, то, как обычно, операции выполняются в следующем порядке:

- 1) возведение в степень,

- 2) умножение и деление,
- 3) сложение и вычитание.

Кроме арифметических операций, выражение фортрана подобно алгебраическому выражению может содержать обращение к функциям. Например, алгебраическая формула

$$A = \frac{\pi}{2c} e^{-bc} \cos(p\sqrt{a^2 - b^2})$$

означает, что для вычисления величины A по известным значениям параметров a , b , c и p нужно:

- 1) вычислить величину $z_1 = a^2 - b^2$;
- 2) вычислить $z_2 = \sqrt{z_1}$, воспользовавшись каким-либо алгоритмом вычисления квадратного корня или таблицами;
- 3) вычислить $z_3 = \cos(pz_2)$, например, по таблицам;
- 4) вычислить $z_4 = e^{-bc}$ опять с помощью какого-либо алгоритма или по таблицам и, наконец,
- 5) умножив и разделив, найти результат

$$A = \frac{\pi}{2c} \cdot z_4 \cdot z_3.$$

В фортране алгоритмы вычисления некоторых функций заложены в самом трансляторе. Полный перечень этих функций, называемых *библиотечными*, дан в приложении 2. Каждая функция имеет свое название — *идентификатор*. В таблице 1 приведены идентификаторы элементарных функций.

Аргумент функции указывается в скобках после идентификатора функции. Для перечисленных ниже функций аргумент должен быть вещественным.

Т а б л и ц а 1

Функция	Идентификатор	Функция	Идентификатор
Квадратный корень	SQRT	Натуральный логарифм	ALOG
Экспонента	EXP	Десятичный логарифм	ALOG 10
Синус *)	SIN	Абсолютная величина	ABS
Косинус *)	COS		
Арктангенс	ATAN		

*) Угол должен быть выражен в радианах.

Например, выражение

SIN(ALFA)

приведет к вычислению синуса угла ALFA. В качестве аргумента допускаются и более сложные комбинации. Так, для вычисления $\sqrt{b^2 - 4ac}$ можно написать

SQRT(B ** 2 - 4.0 * A * C)

Константу, переменную, функцию или комбинацию констант, переменных и функций, разделенных знаками арифметических операций и скобками, называют *арифметическим выражением*. Арифметическое выражение указывает машине, какие операции нужно провести над числами, чтобы получить результат.

В арифметическом выражении всегда

1) число открывающих скобок должно равняться числу закрывающих скобок,

2) два знака арифметических операций не должны стоять рядом.

Примеры арифметических выражений.

A ** 2 * B/C + 1.0

2. * SIN((X + Y)/2.) * COS((X - Y)/2.)

(COS(2.0 * X) + 1.0)/2.0

ALOG((1.0 + X ** 2)/(1.0 - X ** 2))/4.0

PI/(2. * C) * EXP(-B * C) * COS(P * SQRT(A ** 2 - B ** 2))

Упражнения. Запишите на фортране следующие алгебраические выражения:

$$a + bx + cx^2,$$

$$a + x(b + cx),$$

$$(a + bx)^3,$$

$$\frac{a + bx}{c + dx},$$

$$\sqrt{\frac{1 - \cos x}{2}},$$

$$\sqrt{p(p-a)(p-b)(p-c)}$$

$$\ln \left| \frac{1-x}{1+x} \right|.$$

§ 5. Оператор присваивания

Мы привыкли к тому, что речь, посредством которой люди общаются друг с другом, состоит из предложений. Каждое предложение представляет собой законченную мысль, выраженную словами и оформ-

ленную согласно правилам нашего языка. Совокупность правил, по которым строятся предложения, называется синтаксисом. Точно так же и программа решения задачи, записанная на фортране, состоит из последовательности предложений. Предложения в машинных языках обычно называют *операторами*. Каждое предложение-оператор есть законченная инструкция, программа действий для машины.

Простейшим и наиболее употребимым оператором является *оператор присваивания*. Он позволяет присвоить переменной результат вычисления выражения. Запись этого оператора имеет вид:

$$A = E$$

где A — идентификатор, E — арифметическое выражение.

Например, в результате выполнения оператора

$$ABX = A + B * X$$

значение переменной ABX станет равным значению выражения $A + B * X$.

Важно понимать, что символ $=$ в операторе присваивания не имеет смысла математического знака равенства. Например, оператор

$$SUMM = SUMM + 2.5$$

означает следующее. Текущее значение переменной $SUMM$ будет найдено машиной в памяти по идентификатору. К этому значению машина прибавит число 2,5. Описанные два шага называются вычислением арифметического выражения, стоящего справа от знака присваивания. Наконец, результат засылается в ячейку памяти с названием $SUMM$, причем хранящееся первоначально там значение, естественно, будет «забыто» машиной.

Примеры. Правильная запись оператора присваивания

$$DISCR = (B ** 2 - 4.0 * A * C) ** .5$$

$$WATTS = VOLTS * AMPS$$

$$E = 2.71828$$

$$W1 = EXP(0.5) - Y ** 2$$

Неправильная запись оператора присваивания

— $A = -1 * A$ — это верное равенство в алгебраическом смысле ($-a = -1 \cdot a$), но это не есть оператор присваивания, поскольку A не идентификатор;

$(A - B) * (A + B) = A ** 2 - B ** 2$ — как и в предыдущем примере, слева от знака присваивания должен стоять идентификатор переменной, а не выражение;

$A1 = A2 = 0.0$ — нельзя в одном операторе производить два присваивания;

$V = 15A + GAMMA$ — выражение составлено неверно, поскольку 15A не может служить идентификатором.

Упражнения. Запишите на фортране следующие равенства:

$$1. \quad P7 = \frac{1}{16} (429x^7 - 693x^6 + 315x^5 - 35x).$$

$$2. \quad y = a + \frac{b}{x} + \frac{c}{x^2}. \quad 3. \quad y = \frac{ax^2 + bx + c}{x^2}.$$

$$4. \quad f = \frac{1}{ax^2 + b}. \quad 5. \quad x = \frac{1}{d - b} \ln \left(-\frac{ab^2}{cd^2} \right).$$

$$6. \quad p = a \cdot e^{-\frac{b^2 + 2c}{4c}}. \quad 7. \quad u = e^{-ax} \sin(\omega x + \varphi).$$

§ 6. Запись программы на бланках

Программа на фортране записывается на бланках такого вида, как показано на рис. 1. Бланк разграфлен на строки, а каждая строка размечена вертикальными штрихами на 72 позиции. Восемь позиций, нумеруемые как 72—80, вынесены в заголовок бланка. Их назначение будет объяснено ниже. Сверху дана нумерация позиций. Каждый символ должен быть записан в одной из позиций, так что всего на строке с учетом заголовка умещается не более 80 символов. Текст одной строки набивается на одну перфокарту.

Перфокарта (рис. 2) имеет 80 колонок (позиций) и 12 строк. Позиции, так же как и на бланке, нумеруются слева направо. Строки нумеруются сверху вниз. Самая верхняя (ненадписанная) строка имеет номер 12, или называется строкой +. Следующая (также ненадписанная) строка имеет номер 11, или называется строкой —. Затем следуют строки 0, 1, 2, ..., 9. В нулевой строке в каждой позиции на перфокарте напечатан ноль, в первой строке в каждой позиции напечатаны единица и т. д.

21-000000

Номера поездов (пунктов)

Номера строк

44444444

Рис. 2. Перфокарта с поволокой набивкой.

Существуют два способа пробивки символов на перфокартах: в колонку и в строку. При набивке в колонку (см. рис. 2) каждый символ пробивается комбинацией дырочек в одной колонке перфокарты, так что в процессе пробивки перфокарта движется справа налево. Одному символу отвечает одна или несколько дырочек в различных строках одной колонки. Например, цифры набиваются одной дырочкой в соответствующей строке: 0 набивается одной дырочкой в нулевой строке, 1 одной дырочкой в первой строке и т. д. Буквы и другие символы набиваются двумя или тремя дырочками (см. приложение 1). Программисту не надо знать кодировку символов. Специальное печатающее устройство надписывает карту в процессе пробивки. Например, на перфокарте, представленной на рис. 2, содержится текст

DISCR=SQRT(B**2-4.*A*C)

расположенный согласно записи на бланке, показанном на рис. 1. Первые шесть позиций пустые или, как говорят иначе, заполнены пробелами. В седьмой позиции и на бланке, и на перфокарте стоит символ D. Из рис. 2 видно, что символ D кодируется двумя дырочками: одна дырочка пробивается в двенадцатой, самой верхней строке, а другая в четвертой строке. Каждый символ занимает одну позицию, так что последний символ — закрывающая скобка стоит в двадцать девятой позиции.

Отечественные устройства подготовки перфокарт (УПП) набивают символы в строку, причем на один символ отводится восемь позиций. Код УПП дан в приложении 1. На рис. 3 показана перфокарта, на которой в коде УПП пробито то же предложение, что и на перфокарте рис. 2. Отметим, что при построчной пробивке пробелу соответствует отсутствие пробивки в позиции, а при построчной пробивке пробелу соответствует символ (корытце). Проверим текст, пробитый на перфокарте рис. 3. В первых восьми позициях первой строки пробита комбинация 1000 1111 (единица — дырочка, нуль — отсутствие дырочки) — это символ (см. приложение 1), т. е. пробел. Тем же символом заполнены позиции 9—16, 17—24, 25—32, 33—40 и 41—48 первой строки. В позициях 49—56

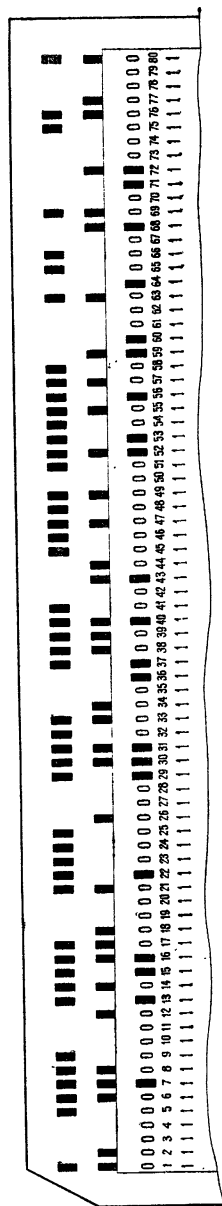


Рис. 3. Перфокарта с построной набивкой.

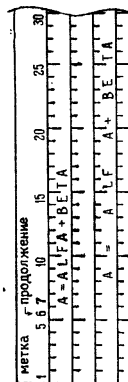


Рис. 4. При записи операторов фортрана пробелы игнорируются.

пробита комбинация 1011 1111 — символ D. Объединя пробивку в следующих восьми позициях с 57 по 64, получим код символа 1:1100 0010. В позициях с 65 по 72 пробит код 1100 1000 — символ S, а в позициях с 73—80 содержится код 0011 0001 — символ C. Таким образом, в первой строке карты отперфорировано содержимое первых десяти позиций строки бланка:

— — — — — DISC

Содержимое полной строки бланка максимально занимают восемь строк на перфокарте. Вне зависимости от того, как пробивается информация на перфокартах построчно или в колонку), одна строка бланка пробивается на одну перфокарту с учетом номера позиции каждого символа.

Операторы фортрана записываются на бланке в поле оператора, которое занимает позиции с 7 по 72 включительно. Оператор может быть записан в любом месте этого поля, поскольку пробелы между символами транслятор игнорирует. Обычно программисты оставляют пробелы таким образом, чтобы программу было удобно считать, но делать это вовсе не обязательно.

Пример. Два оператора на рис. 4 эквивалентны.

В фортране нет специальных символов, указывающих начало и конец оператора. Считается, что запись оператора начинается с новой строки и начало поля оператора служит одновременно началом самого оператора. Другими словами, в фортране, за исключением случаев, о которых мы скажем ниже, каждый оператор записывается на отдельной строке и пробивается на отдельной перфокарте. Такой способ записи делает текст программы очень легким для чтения, а колоду перфокарт удобной для внесения изменений.

Если оператор настолько длинен, что он не помещается на одной строке и соответственно на одной перфокарте, его можно перенести на следующую строку (перфокарту), поставив признак продолжения. Признаком того, что данная строка (перфокарта) является продолжением предыдущей, служит любой

Позиции 72 — 80 также не обрабатываются транслятором. Эти позиции обычно используются для пробивки в них названия программы и номера перфокарты.

Пример. На рис. 5 приведен участок программы вычисления функции Бесселя. У этой программы в строках со второй по шестую записан комментарий. Обратите внимание на то, что текст комментария для наглядности отделен от операторов пустыми строками. Операторы, записанные в девятой и десятой строках, имеют метки 100 и 110 соответственно. Метки записываются и пробиваются на картах в позициях с первой по пятую. Метками в языке фортран служат числа. За оператором с меткой 110 следует оператор присваивания, не уместившийся в одной строке. Строки продолжения имеют символы (переноса) 1, 2 и 3 в шестой позиции.

§ 7. Ввод и вывод информации

Обычно цель работы программы состоит в вычислении значений одной или нескольких величин. Чтобы программист мог узнать результат расчета, он должен каким-либо образом вывести информацию из памяти машины. Простейшим способом вывода является печать на бумажной ленте. Печать информации ведется построчно. Автоматическое цифровое печатающее устройство (АЦПУ) набирает и печатает сразу строку, состоящую из 128 символов, включая пробелы. Затем бумага сдвигается на интервал, печатается следующая строка и т. д. Программист может по своему усмотрению располагать выводимую информацию в 128 позициях строки бумаги. Например, если нужно отпечатать значения ста переменных, то эти сто чисел можно выводить в столбец по одному числу на строку, начиная, к примеру, с десятой позиции. Вся выдача в этом случае будет состоять из ста строк. Ясно, что такой способ вывода не эффективен с точки зрения расхода бумаги, потому что большая ее площадь останется чистой. Экономнее располагать на строке несколько чисел. Если, например, печатать по десять чисел на строке, то под каждое из них можно отвести по двенадцать позиций, что обычно бывает достаточно.

В фортране печать значений переменных осуществляется оператором PRINT *).

*) От английского слова to print — печатать.

Этот оператор указывает:

- 1) значения каких переменных нужно отпечатать,
- 2) каким образом информация должна располагаться на строке.

Описание расположения информации на бумаге может быть довольно сложным, и для того, чтобы не перегружать оператор PRINT, оно вынесено в специальную конструкцию, которая вызывается оператором FORMAT. Оператор FORMAT обязательно имеет метку, посредством которой на него можно ссылаться в операторах ввода/вывода.

В общем случае оператор печати информации имеет вид

$$\text{PRINT } n, P_1, P_2, \dots P_k$$

где n — метка оператора FORMAT, описывающего нужное программисту расположение информации на бумаге, а P_1, P_2, \dots, P_k — список переменных, т. е. перечисленные через запятую идентификаторы переменных, значения которых нужно отпечатать.

Примеры записи оператора вывода.

1) PRINT2, RHO

По этому оператору машина отпечатает значение переменной RHO, причем число будет расположено на строке согласно описанию, данному в операторе FORMAT с меткой 2.

2) PRINT71, XMAX, N, DX

По этому оператору машина отпечатает значения трех переменных. Первым будет напечатано значение переменной XMAX, вторым значение переменной N, третьим значение DX. Расположение чисел на строке определяется оператором FORMAT с меткой 71.

Если для работы программы требуется задание значения одной или нескольких переменных, то ввод этих значений можно произвести с перфокарт. Перед работой программы перфокарты, на которых набиты нужные числа, закладываются в специальное читающее устройство машины. В языке фортран имеется оператор READ *) — оператор чтения с перфокарт. Поэтому оператору машина обращается к читающему устройству и вводит в свою память на указанные места пробитую на картах информацию.

*) От английского слова to read — читать.

Оператор READ подобно оператору печати указывает машине:

1) значения каких переменных должны быть введены с перфокарт в память машины;

2) каким образом, т. е. в каких позициях и в какой форме, набиты эти значения.

Информация может быть пробита на перфокартах различными способами. Например, одна перфокарта может содержать одно или несколько чисел. Способ пробивки вводимой информации задается оператором FORMAT.

В общем случае оператор чтения с перфокарт имеет вид

$$\text{READ } n, P_1, P_2, \dots, P_k$$

где n — метка оператора FORMAT, в котором описывается расположение информации на перфокартах, а P_1, P_2, \dots, P_k — список переменных, которым должны быть присвоены вводимые значения.

Примеры: записи оператора ввода.

1) READ 90, U

Встретив такой оператор, машина обратится к читающему устройству, прочтает с перфокарты одно число, пробитое как описано в операторе FORMAT с меткой 90, и присвоит это значение переменной U.

2) READ 999, XMASS, PMASS

По этому оператору машина прочтает с перфокарт два числа, значение первого из них присвоит переменной XMASS, а значение второго переменной PMASS. Оператор FORMAT с меткой 999 указывает способ пробивки чисел на перфокартах.

Оператор ввода/вывода информации посредством метки ссылается на оператор FORMAT, указывающий, каким образом записаны программистом вводимые числа и в какой форме машина должна печатать выводимые числа. Полное описание оператора FORMAT будет дано во второй части книги, а здесь мы ограничимся простейшими способами ввода/вывода.

Как уже говорилось в §§ 2 и 3, числа могут храниться в машине как целые и как вещественные. Разные типы чисел вводятся и выводятся по-разному. Для указания того, какого типа величина должна

быть введена/выведена машиной, в фортране имеются спецификации оператора FORMAT. Рассмотрим три из них.

Для ввода/вывода значений целых величин служит спецификация I, имеющая вид

In

где I — буква, служащая признаком типа спецификации, а n — целое число, указывающее, сколько позиций на перфокарте или на строке бумажной ленты отводится под запись значения целой величины.

При вводе по спецификации I число, пробитое в n позициях перфокарты, рассматривается машиной как вводимое значение переменной. Числу может предшествовать знак плюс или минус. Если знак у числа отсутствует, оно считается положительным. Запись числа в n позициях перфокарты должна соответствовать записи целой константы (см. § 2), т. е. не должна содержать десятичной точки. Пробелы между цифрами игнорируются.

Пример. Нужно ввести с перфокарт в память машины значения переменных KAPPA и KSI, равные соответственно —6 и 531. Числа пробиты на перфокарте как показано на рис. 6, т. е. первое число вместе с пробелами занимает три позиции, а второе число пять позиций. Мы пишем

READ 15, KAPPA, KSI

15 FORMAT (I3, I5)

Оператором FORMAT программист может по своему усмотрению перераспределять роль позиций. Так, если вместо выписанного выше оператора поставить

15 FORMAT (I4, I3)

то переменной KAPPA будет и присвоено значение —6 (число в первых четырех позициях на перфокарте), в переменной KSI значение 53 (число в следующих трех позициях).

Аналогичным образом работает спецификация I при выводе. На строке под число отводится поле ши-

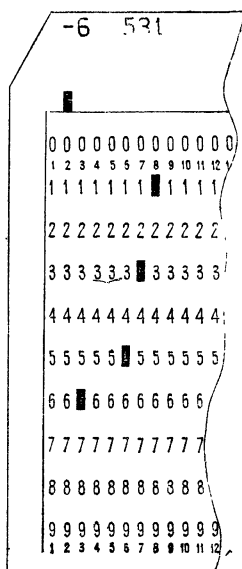


Рис. 6. Набивка по спецификации I.

риной в n позиций, в котором оно печатается в виде целой константы. Если ширина поля превосходит количество цифр в числе, то число печатается в правой части поля, а оставшиеся левые позиции заполняются пробелами.

Пример. Если переменные KAPPA и KSI имеют значения, указанные в предыдущем примере, то по приказу

PRINT 16, KAPPA, KSI

16 FORMAT (110, 15)

машина печатает строку

$\underbrace{\hspace{8em}}_{8 \text{ пробелов}} \quad -6 \quad \underbrace{\hspace{12em}}_{12 \text{ пробелов}} \quad 531$

Число -6 стоит на строке в девятой и десятой позициях, а число 531 занимает позиции с 23 по 25.

Значения вещественных переменных можно вводить и выводить по двум спецификациям F и E, связанным с двумя формами записи вещественных констант. По спецификации F вещественная константа записывается без десятичного порядка, например 0.017453. Форма записи числа с десятичным порядком, например запись 1.7453E-2, отвечает спецификации E.

Спецификация F имеет вид

$F_n \cdot m$

где F — буква, служащая признаком типа спецификации, n — целое число, указывающее, сколько позиций отводится под запись числа, m — число позиций, отводимых под дробную часть числа.

При вводе значения переменной спецификации F оно должно быть пробито в n позициях перфокарты в виде вещественной константы. Машина сама отыскивает в записи числа десятичную точку, и поэтому параметр m при вводе игнорируется. При выводе по спецификации F значение вещественной переменной печатается в правой части поля строки длиной в n позиций. Из них m позиций занимает дробная часть числа, одна позиция отводится под десятичную точку и одна под знак числа. Знак плюс перед числом не печатается.

Пример. Число — 273,16 пробито на перфокарте как показано на рис. 7. Пусть это число является начальным значением переменной ZERO. Рассмотрим работу операторов

READ 1, ZERO

PRINT 2, ZERO

1 FORMAT (F10.3)

2 FORMAT (F15.5)

Выполняя оператор READ, машина прочитает одну перфокарту и расшифрует ее содержимое, руководствуясь спецификацией оператора FORMAT с меткой 1. Спецификация указывает, что значение вещественной переменной пробито в первых десяти позициях перфокарты. Машина возьмет содержащееся в этих позициях число — 273.16 и присвоит его переменной ZERO.

По оператору PRINT значение ZERO будет отпечатано на бумаге в виде, определяемом оператором FORMAT с меткой 2, т. е. в первых пятнадцати позициях строки с пятью знаками после десятичной точки:

— 273.16000

↑
15-я позиция

Если при составлении программы заранее неизвестно, каков может быть порядок вводимых и выводимых величин, то удобнее пользоваться спецификацией E, имеющей вид

$E_n.m$

где E — буква, служащая признаком типа спецификации, n — целое число, указывающее, сколько позиций отводится под запись числа, m — число позиций, отводимых под дробную часть числа.

Ввод по спецификации E аналогичен вводу по спецификации F. Последовательность символов, стоящая в n позициях перфокарты, рассматривается как запись вещественной константы, значение которой присваивается переменной. При выводе числа по

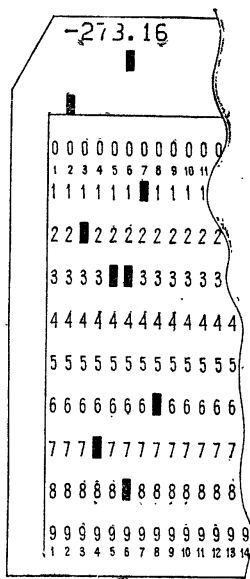


Рис. 7. Набивка по спецификации F.

спецификации E оно печатается в виде

$$\begin{array}{c} \pm X \\ \text{1 цифра,} \\ \text{целая} \\ \text{часть} \end{array} . \underbrace{XX \dots X}_{\text{m цифр,} \\ \text{дробная часть}} \pm \underbrace{ee}_{\text{2 цифры,} \\ \text{порядок}}$$

Число располагается в правых позициях отводимого под него поля. Одну позицию занимает знак числа (знак плюс опускается), за ним печатается одна цифра (не ноль) перед десятичной точкой, потом следует десятичная точка и *m* позиций за ней отводятся под дробную часть числа. После дробной части печатается знак порядка и две цифры порядка числа. Отметим, что на некоторых машинах формат печати по E-спецификации несколько отличается от описанного, принятого в Фортране-Дубна. Эти отличия мы поясним примером. В дубненской версии число —36,288 по спецификации E 12.3 печатается в виде —3.629 + 01. В других вариантах транслятора та же спецификация может дать формы записи вида —.363 + 02 либо —3.629E 01.

Примеры ввода/вывода по спецификации E.

Пусть в программе встречается последовательность операторов:

```
READ 14, ONE, TWO
14 FORMAT (E10.3, E7.2)
```

Оператор FORMAT указывает, что значение переменной ONE должно быть пробито в первых десяти позициях перфокарты, а значение переменной TWO в позициях с 11 по 17. Например, как показано на рис. 8. После работы оператора READ величина ONE равна $5,8121 \cdot 10^4$, а величина TWO равна $5 \cdot 10^{-2}$. Отпечатаем эти числа:

```
PRINT 15, ONE, TWO
15 FORMAT (E15.3, E15.2)
```

Выдача имеет вид

```
5.812 + 04      5.00 - 02
      ↑          ↑
    15-я позиция 30-я позиция
```

Под каждое число отводится по пятнадцати позиций на строке бумаги.

Последовательность идентификаторов, перечисленных в операторе ввода/вывода, называется *спис-*

ком ввода/вывода, а последовательность спецификаций в операторе FORMAT — *списком спецификаций*. Таким образом, операторы ввода/вывода и оператор FORMAT имеют вид

READ n , <список ввода>

PRINT n , <список вывода>

n FORMAT (<список спецификаций>)

где n — метка.

Следует иметь в виду, что длина списков операторов ввода/вывода и оператора FORMAT, по которому

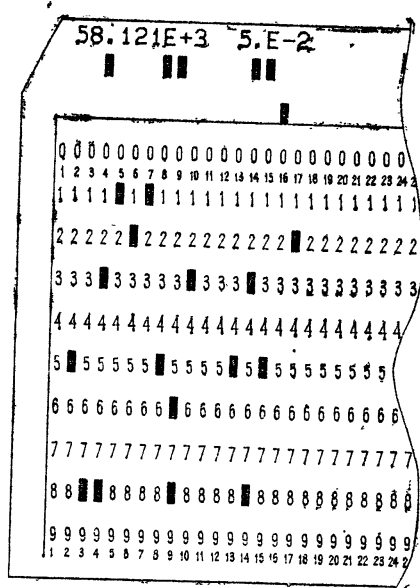


Рис. 8. Набивка по спецификации E.

происходит ввод/вывод, может не совпадать. Например, мы можем написать

PRINT 2010, P, RHO, U
2010 FORMAT (F20.5)

Здесь список вывода содержит три переменные, а список оператора **FORMAT** — одну спецификацию. Каким образом в этом случае печатаются значения переменных легко понять, если учесть разницу в функциях списков ввода/вывода и списков спецификаций. Список ввода/вывода перечисляет переменные, значения которых должны быть введены в память машины или выведены из нее. Список спецификаций указывает, каким образом входная или выходная информация расположена на перфокарте или листе бумаги. В предыдущем примере машина печатает значения трех величин: P , RHO и U . Оператор **FORMAT** указывает, что на строке располагается одно число, что под него отводятся первые двадцать позиций и что оно имеет вид, предписываемый спецификацией F . Поэтому значения P , RHO и U печатаются друг под другом на трех строках в первых двадцати позициях каждое. Так, если P , RHO и U равны 88,72609; 0,0121996 и $-1850,94$ соответственно, то печать имеет вид

88.72609
0.01220
— 1850.94000
<div style="display: inline-block; width: 100px; text-align: center;"> <div style="display: flex; justify-content: space-around; width: 100%;"> ↑ ↑ </div> <div style="display: flex; justify-content: space-around; width: 100%;"> 15-я позиция 20-я позиция </div> </div>

Упражнения.

В следующих упражнениях предполагается, что значения вводимых величин пробиты по одному числу на перфокарте по спецификации E15.3. Для вывода используйте спецификацию E20.3.

Напишите участки программы, которые выполняют следующие операции:

- ① а) Ввести значения переменных a , b , c .
- б) Вычислить

$$X1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a},$$

$$X2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

- в) Напечатать значения a , b , c , $X1$, $X2$.
2. а) Ввести значения переменных r , h .
- б) Вычислить

$$a = 2\sqrt{2hr - h^2}.$$

- в) Напечатать значения r , h , a .

3. а) Ввести значения переменных R, H1, H2.
 б) Вычислить

$$S = \pi R \left(H1 + H2 + R + \sqrt{R^2 + \left(\frac{H2 - H1}{2} \right)^2} \right),$$

$$V = \pi R^2 \frac{H1 + H2}{2}.$$

- в) Напечатать значения R, H1, H2, S, V.
 4. а) Ввести значение переменной x.
 б) Вычислить

$$y = \operatorname{arctg} \frac{\sqrt{1 - x^2}}{x}.$$

- в) Напечатать значения x, y.
 5. а) Ввести значения переменных x, a.
 б) Вычислить

$$u = \ln \frac{(x + a)^3}{x^2 - ax + a^2},$$

$$v = \operatorname{arctg} \frac{x \sqrt{3}}{2a - x},$$

$$w = \frac{1}{2} u + \sqrt{3} v.$$

- в) Напечатать значения x, a, u, v, w.

§ 8. Оформление программы

Ознакомившись с оператором присваивания и операторами ввода и вывода информации, мы можем составить не очень сложную программу.

Как уже говорилось во введении, программа — это алгоритм решения задачи, записанный на некотором языке. Она состоит из последовательности операторов, которые машина должна выполнить для решения данной задачи. В языке фортран каждая программа оформляется по определенным правилам. Первым оператором программы, указывающим ее начало и имя, служит оператор PROGRAM, имеющий вид

PROGRAM P

где P — идентификатор, имя программы. Как и любой идентификатор, он может содержать до шести

буквенно-цифровых символов, первым из которых должна быть буква.

Примеры записи оператора PROGRAM.

```
PROGRAM LINE  
PROGRAM A007  
PROGRAM HECUBA
```

Следует иметь в виду, что в некоторых версиях языка (например, в фортране IV) оператор PROGRAM отсутствует и за начало программы принимается ее первый оператор.

Последним из операторов, относящихся к программе, должен стоять оператор *)

END

указывающий на окончание последовательности операторов, составляющих данную программу. Таким образом, к программе относятся все операторы, заключенные между PROGRAM и END.

Приступив к счету по программе, машина выполняет один за другим все содержащиеся в ней операторы, начиная с первого. В упрощенном виде схема счета задачи состоит в следующем. Машина обращается к первому оператору программы (как говорят, передает ему управление) и делает все указанные в нем операции. Например, если это оператор присваивания, то машина по идентификаторам переменных находит их значение в своей памяти, производит над числами операции, имеющиеся в выражении, которое стоит справа от знака присваивания, а после вычисления выражения присваивает его значение переменной, как указано в операторе. Выполнив первый оператор, машина выполняет все действия, предписываемые вторым оператором, затем переходит к следующему оператору и т. д. Ясно, что на каком-то этапе счет программы должен быть закончен. Место, дойдя до которого машина должна закончить счет программы, указывается оператором

STOP

*) От английского слова end — конец.

Важно понимать, что операторы STOP и END имеют разные функции. Оператор END есть указание транслятору, т. е. программе, переводящей данную программу на язык машинных кодов, что операторы, относящиеся к данной программе, на этом месте заканчиваются. Далее, например, может следовать текст другой программы. Оператор STOP есть указание машине, что при выполнении программы дальше этого места считать не нужно. Правда, зачастую операторы STOP и END стоят рядом друг с другом (конечно, STOP должен стоять перед END), но бывают и ситуации, когда оператор STOP встречается в середине программы. Например, на каком-то этапе счета мы проверяем выполнение некоторого условия. Если это условие выполнено, мы продолжаем работу программы, а при невыполнении условия счет нужно прекратить, т. е. передать управление на оператор STOP. Каким образом записываются на языке фортран условия такого рода, мы объясним дальше. Здесь же для нас важно, что оператор STOP может оказаться внутри программы, а после него могут идти операторы, работающие в случае выполнения некоторого условия.

Пример. Пусть $(x_1, y_1), (x_2, y_2)$ — две точки на плоскости. Тогда

$$\frac{y - y_2}{x - x_2} = \frac{y_1 - y_2}{x_1 - x_2}$$

является уравнением прямой, проходящей через эти точки. На рис. 9 приведена программа вычисления ординаты y точки, лежащей на прямой, по ее абсциссе x .

МЕТКА	ПРОДОЛЖЕНИЕ
1	5 6 7 10 15 20 25 30
	PROGRAM LINE
C	
C	ВВОД ЗНАЧЕНИЙ ПАРАМЕТРОВ
C	
	READ 1, X1, Y1
	READ 1, X2, Y2
	READ 1, X
1	FORMAT (F10.3, E10.3)
C	
C	ВЫЧИСЛЕНИЕ ОРДИНАТЫ Y
C	
	ALFA = (Y1 - Y2) / (X1 - X2)
	Y = Y2 + ALFA * (X - X2)
C	
C	ПЕЧАТЬ КООРДИНАТ ТОЧКИ
C	
	PRINT 1, X, Y
	STOP
	END

Рис. 9. Программа вычисления ординаты точки, лежащей на заданной прямой.

§ 9. Операторы передачи управления

До сих пор мы считали, что машина выполняет программу последовательно, оператор за оператором, пока не дойдет до оператора STOP. Такой принцип работы не всегда удобен. В программе может потребоваться обойти или повторить некоторую последовательность операторов в зависимости от выполнения или невыполнения некоторых условий. Так, например, при вычислении корней квадратного уравнения следует проверять, положителен или отрицателен дискриминант уравнения, и в каждом случае действовать по-разному.

В фортране имеется несколько операторов передачи управления, позволяющих менять естественный процесс выполнения программы.

9.1. Оператор безусловной передачи управления. *Безусловная передача управления* из одного места программы в другое производится оператором *)

GO TO n

где n — метка оператора, который должен выполняться следующим.

Пример. Если участок программы имеет вид

```
.....  
XX = ABS (XXX)  
X=0.125 * XX  
GO TO 10  
20.F=4.0 * X ** 2-2.0  
.....  
10 X=1.0/X
```

то следующим после оператора $X = 0.125 * XX$ выполняется оператор $X = 1.0/X$, а не оператор $F = 4.0 * X ** 2 - 2.0$, как это было бы при отсутствии оператора передачи управления.

9.2. Арифметический оператор условной передачи управления. Этот оператор также нарушает естественный порядок выполнения программы, но управление может быть передано в три различных места программы в зависимости от того, меньше, равно или больше нуля некоторое арифметическое выражение.

*) Английское сочетание go to переводится как «иди к».

Арифметический оператор IF)* имеет вид

IF (E) n_1, n_2, n_3

где E — арифметическое выражение, n_1, n_2, n_3 — метки операторов. Программа переходит к выполнению оператора с меткой n_1 , если выражение E отрицательно, к оператору с меткой n_2 , если выражение E равно нулю, и к оператору с меткой n_3 , если выражение E положительно.

Пример. При вычислении корней квадратного уравнения $ax^2 + bx + c = 0$ необходимо проверить, положителен ли дискриминант $b^2 - 4ac$ этого уравнения. Такая проверка может быть выполнена операторами

DISCR = B ** 2 - 4.0 * A * C

IF (DISCR) 11, 12, 12

Оператор IF указывает, что при отрицательном дискриминанте следующим будет выполняться оператор с меткой 11. Если же дискриминант положителен или равен нулю, то управление будет передано на оператор с меткой 12.

9.3. Оператор цикла. С помощью оператора безусловной передачи управления и оператора IF можно организовать циклы. Циклом называется последовательность операторов, которая выполняется несколько раз в процессе счета программы при различных значениях некоторой переменной.

Пример. Пусть нужно вычислить факториал числа N, т. е. произведение $1 \cdot 2 \cdot 3 \dots N$. Обозначив результат через NI, напомним следующий участок программы:

NI = 1

I = 1

21 NI = NI * I

22 IF(I — N) 23, 25, 25

23 I = I + 1

24 GO TO 21

25 1

К моменту перехода на оператор с меткой 25 значение переменной NI равно факториалу числа N. Операторы с метками 21, 22, 23, 24 образуют цикл, т. е. последовательность операторов,

*) От английского слова if — если.

которая выполняется несколько раз (в данном случае N раз) при различных значениях некоторой переменной (в данном случае переменной I).

В фортране имеется специальный оператор, упрощающий организацию циклов. *Оператор цикла DO* *) имеет вид

$$DO\ n\ i = m_1, m_2, m_3$$

где n — метка, i — целая переменная, m_1, m_2, m_3 — целые константы без знака или целые переменные.

Оператор DO служит первым оператором цикла. Последним операторам цикла является оператор с меткой n . Оператор DO позволяет выполнить несколько раз все операторы, заключенные между ним и оператором с меткой n включительно при различных значениях переменной i . Целая переменная i служит счетчиком цикла. Начальное значение этой переменной равно m_1 . Цикл продолжает выполняться до тех пор, пока значение переменной i не станет больше m_2 . Таким образом, m_2 — конечное значение переменной цикла. Величина m_3 определяет шаг, с которым меняется счетчик i .

Пример. С помощью оператора цикла DO участок программы для вычисления факториала числа N может быть записан следующим образом:

```
NI = 1
DO 10 I = 1, N, 1
10 NI = NI * I
```

Здесь I — счетчик цикла, 1 — начальное значение счетчика, N — конечное значение счетчика. Изменение I происходит с шагом единица.

Оператор цикла DO указывает на то, что все следующие за ним операторы вплоть до оператора с меткой n должны быть выполнены сначала при значении счетчика i , равном m_1 . Затем значение счетчика становится равным $m_1 + m_3$, и выполнение цикла повторяется. Затем цикл выполняется при $i = m_1 + 2m_3$ и т. д. до тех пор, пока значение i не превзойдет значение m_2 . Если значение счетчика i станет большим значения m_2 , выполнение цикла прекращается и машина переходит к выполнению оператора,

*) От английского глагола to do — делать.

следующего непосредственно за оператором с меткой n .

Если шаг цикла m_3 равен единице, то величину m_3 можно опустить и писать, например,

DO 10 I = 1, N

При записи оператора цикла необходимо выполнять некоторые правила:

1) величины m_1 , m_2 и m_3 могут быть только целыми константами без знака или целыми простыми переменными,

2) переменная счетчика цикла i может принимать только положительные значения (в частности, цикл нельзя начинать с нуля),

3) шаг цикла не может быть отрицательным.

Примеры правильной записи оператора DO:

DO 101 IMAGE = INDEX, K, KAPPA

DO 57 L = 12, 17

DO 322 LAMDA = 1, N, 8

Примеры неправильной записи оператора DO:

DO 86 I = 0, 50

— цикл не может начинаться с нуля;

DO 93 MATRIX = 50, 5, -1

— шаг цикла не может быть отрицательным;

DO 326 K = 2, A, 3

— параметры цикла должны быть целыми величинами;

DO 45 I = N, MAX - 1

— выражения в записи оператора цикла не допускаются.

После выполнения последнего оператора цикла DO машина должна перейти к выполнению первого оператора цикла, либо выйти из цикла, в зависимости от значения счетчика. Поэтому последний оператор цикла не должен быть оператором передачи управления. В то же время встречаются программы (см. пример ниже), в которых при соблюдении этого правила нечем закончить цикл, т. е. нет оператора, которому можно приписать метку и который можно поставить последним оператором цикла DO. Выход из такой ситуации состоит в использовании оператора *) CONTINUE.

*) От английского слова continue — продолжение.

Оператор CONTINUE является «пустым» оператором. Он не указывает на выполнение каких-либо команд, но он может иметь метку и таким образом указывать место конца цикла или место, куда передается управление.

Пример. Напишем программу вычисления количества «счастливых» билетов. Шестизначный номер билета (автобусного, троллейбусного, трамвайного) считается «счастливым», если сумма первых трех его цифр равна сумме последних. Так, номер 451 802 счастливый, потому что $4 + 5 + 1 = 8 + 0 + 2$. В программе нужно устроить перебор всех номеров билетов, проверяя для каждого равенство сумм первых и последних цифр. Если

метка	продолжение
1	5 6 7 10 15 20 25 30
	PROGRAM TICKET
C	
C	ЧИСЛО СЧАСТЛИВЫХ БИЛЕТОВ
C	
	LUCK = 0
	DO 1 I = 1, 9
	DO 1 J = 1, 9
	DO 1 K = 1, 9
	DO 1 L = 1, 9
	DO 1 M = 1, 9
	DO 1 N = 1, 9
	IF (I + J + K - L - M - N) 1, 2, 1
2	LUCK = LUCK + 1
1	CONTINUE
	PRINT 3, LUCK
3	FORMAT(15)
	STOP
	END

Рис. 10. Программа расчета количества «счастливых» билетов.

номер «счастливый», то к накопителю числа «счастливых» билетов прибавляем единицу. Каждая из шести цифр номера билета принимает значение от нуля до девяти. Поскольку в фортране цикл не может начинаться с нуля, введем переменные, на единицу превосходящие значения цифр. Итак, пусть I, J, K — увеличенные на единицу значения первых трех цифр билета; L, M, N — увеличенные на единицу значения последних трех цифр билета; LUCK — накопитель числа «счастливых» билетов. Текст программы приведен на рис. 10.

Упражнения.

1. Написать участки программы, выполняющие следующие операции:

а) Имеются три переменные: X_1, X_2, X_3 . Выбрать переменную с максимальным значением и присвоить это значение переменной X_{MAX} .

б) Выбрать из переменных X_1, X_2, X_3 переменную с минимальным значением и присвоить это значение переменной X_{MIN} .

в) Переставить значения переменных X_1, X_2, X_3 , упорядочив их по возрастанию.

г) Найти максимальное целое число, не превосходящее значения переменной X , и присвоить его переменной $INTX$.

2. Угол $THETA$, получающийся в результате работы предыдущей части программы, может быть произвольным. Написать участок программы приведения значения этого угла к интервалу $[0, 2\pi)$.

3. В точке x вычислить значения пятого многочлена Эрмита $H_5(x)$, используя значения

$$H_1(x) = 2x, \quad H_2(x) = 4x^2 - 2$$

и рекуррентное соотношение

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x).$$

4. Метод Ньютона решения уравнения $F(x) = 0$ состоит в следующем. Если x_s — приближительное значение корня уравнения, то следующее приближение находится по формуле

$$x_{s+1} = x_s - \frac{F(x_s)}{F'(x_s)},$$

где через $F'(x)$ обозначена производная функции $F(x)$.

Найти методом Ньютона корень уравнения

$$x = e^{-x}$$

с точностью до 10^{-4} , взяв в качестве первого приближенного значения корня $x = 0$.

§ 10. Массивы переменных

Иногда бывает удобным присваивать имя не одной переменной, а целой группе, или, как говорят в программировании, *массиву* переменных. Так, в алгебре вводятся обозначения для векторов и матриц, а каждый вектор и каждая матрица представляют собой набор (массив) элементов. Для указания конкретного элемента из этого набора используются индексы. Например, первый элемент вектора a обозначают a_1 , второй элемент обозначают a_2 и т. д. Если элементы массива нумеруются одним индексом, то массив называется одномерным. Если же для указания элемента массива нужно большее число индек-

сов, то массив называется двумерным, трехмерным и т. д. по их числу. Обычная матрица, например, является двумерным массивом.

В фортране также возможно давать название массиву переменных. Обращение к конкретной переменной массива состоит в указании идентификатора массива и индексов, определяющих место этой переменной в массиве. Индексы ставятся после идентификатора и заключаются в скобки. Например,

A (40) — сороковой элемент массива A,
MATRIX (I, J) — элемент массива-матрицы MATRIX стоящий в I-й строке и J-м столбце.

В отличие от *переменной с индексами*, переменную без индексов называют *простой переменной*.

Чтобы указать машине, что некоторые идентификаторы являются идентификаторами массивов переменных определенной размерности и определенной длины, все идентификаторы массивов должны быть описаны. Для описания массивов переменных служит оператор DIMENSION*), который имеет вид

DIMENSION <список>

где <список> есть перечисленные через запятую идентификаторы массивов с указанием в скобках наибольшего возможного в программе значения каждого из индексов.

Пример. В программе встречается матрица VERTEX размера 2×4 , вектор PSI длины 21 и трехмерная матрица R размера $10 \times 5 \times 8$. Эти массивы могут быть описаны оператором

DIMENSION VERTEX (2, 4), PSI (21), R (10, 5, 8)

Оператор DIMENSION не указывает на вычисление, а лишь содержит информацию для транслятора о структуре переменных. Он относится к классу так называемых неисполняемых операторов. Оператор DIMENSION должен стоять в самом начале программы и предшествовать первому исполняемому оператору.

*) От английского слова dimension — размеры, объем.

Пример оформления начала программы
 PROGRAM STATE
 DIMENSION P (200), RHO (200)
 READ 10, TE, TI

.

Здесь первым исполняемым оператором является оператор чтения с перфокарт значений переменных TE и TI. Оператор DIMENSION предшествует этому оператору.

На идентификатор переменной с индексом накладываются те же ограничения, что и на идентификатор простой переменной (т. е. переменной без индекса). Идентификатором может служить любая комбинация от одного до шести буквенно-цифровых символов, причем первым символом должна быть буква. Если идентификатор начинается с букв I, J, K, L, M, N, то все переменные массива являются целыми величинами. В противном случае массив состоит из вещественных переменных.

Пример. Оператор

DIMENSION I (10, 15), MAN (181), WHEN (2, 17)

указывает, что I и MAN служат идентификаторами массивов целых переменных, а WHEN идентификатором массива действительных переменных.

Переменные с индексом могут встречаться в арифметических выражениях точно так же, как и простые переменные. Запись

$$SUMM = SUMM + A(I)$$

указывает, что машина должна сложить текущее значение переменной SUMM со значением i -го элемента массива A и присвоить результат той же переменной SUMM.

Размерность массивов в языке фортран не может быть сколь угодно большой. Максимально допустимая размерность зависит от версии языка, но в большинстве версий она равна трем, т. е. допускаются только одномерные, двумерные и трехмерные массивы.

Индекс переменной должен быть целой величиной, поскольку он указывает номер переменной в массиве. Значение индекса, возникающее при счете программы, должно лежать в пределах от единицы

до максимального значения, указанного в операторе DIMENSION. В частности, значение индекса не может быть отрицательным или равняться нулю. Например, если в программе имеется оператор

DIMENSION ABC(10)

то в памяти машины отводится место для хранения значений десяти переменных, к первой из которых можно обращаться как ABC(1), ко второй как ABC(2) и т. д. Переменных ABC(0), ABC(—5) и ABC(11) в машине нет. Обращение к ним в программе приведет к ошибке в вычислениях.

Ввод и вывод значений массива переменных можно производить как полностью, так и по частям. При вводе/выводе значений всего массива в операторах READ и PRINT достаточно указать его идентификатор. Например, если в операторе DIMENSION массив A описан как A(4,4), то по оператору ввода

READ 10, A

машина прочитает с перфокарт шестнадцать чисел.

Последовательность ввода элементов массива определяется следующими правилами. Если массив одномерный, то элементы вводятся в порядке возрастания индекса, т. е. первое число, прочитанное машиной с перфокарты, присваивается первому элементу массива, второе число присваивается второму элементу массива и т. д. Поэтому, если VEC — одномерный массив из трех элементов

DIMENSION VEC(3).

то оператор

READ 11, VEC

эквивалентен оператору

READ 11, VEC(1), VEC(2), VEC(3)

Элементы двумерного массива вводятся по столбцам, если считать их расположенными в виде матрицы. Поэтому операторы

DIMENSION MAT(4,4)

READ 12, MAT

указывают на следующий порядок ввода элементов массива MAT: MAT(1,1), MAT(2,1), MAT(3,1), MAT(4,1), MAT(1,2), MAT(2,2), ..., MAT(3,4), MAT(4,4). Таким образом, быстрее меняется первый индекс массива.

Аналогичное правило действует и для трехмерных массивов: быстрее всего меняется первый индекс, медленнее всего — последний индекс. Поэтому операторы

```
DIMENSION RIX(3, 4, 2)
```

```
.....
```

```
READ 5, RIX
```

```
5 FORMAT(E20.3)
```

эквивалентны следующей последовательности операторов:

```
DIMENSION RIX(3, 4, 2)
```

```
.....
```

```
DO 13 K = 1, 2
```

```
DO 13 J = 1, 4
```

```
DO 13 I = 1, 3
```

```
13 READ 5, RIX(I, J, K)
```

```
5 FORMAT(E20.3)
```

Отдельные элементы массивов можно вводить и выводить подобно переменным без индекса, просто перечисляя их в операторе ввода/вывода. Например, если мы запишем операторы

```
DIMENSION A(4, 4), B(3, 2)
```

```
READ 100, A(1, 2), B(3, 2)
```

то с перфокарт будут прочитаны значения двух переменных: элемента A(1, 2) массива A и элемента B(3,2) массива B.

Массивы переменных удобно обрабатывать в цикле с помощью оператора DO.

Пример. Составим участок программы, производящий упорядочение элементов одномерного массива X длины N, так, чтобы после выполнения этого участка элементы массива располагались по возрастанию: $X(1) \leq X(2) \leq \dots \leq X(N)$.

Алгоритм метода состоит в следующем. Каждый элемент массива $X(I)$, $I = 1, 2, \dots, N-1$ сравниваем со всеми последующими элементами $X(J)$, $J > I$. Если $X(J)$ меньше $X(I)$, элементы $X(I)$ и $X(J)$ меняем местами. Запись этого участка программы приведена на рис. 11.

Сделаем некоторые пояснения относительно текста программы. В операторе цикла DO нельзя записать в качестве пределов

метка	продолжение
1	5 6 7 10 15 20 25 30
	NM1 = N - 1
	DO 5B I = 1, NM1
	IP1 = I + 1
	DO 5B J = IP1, N
	IF (X(I) - X(J)) 5B, 5B, 51
51	ZX = X(I)
	X(I) = X(J)
	X(J) = ZX
5B	CONTINUE

Рис. 11. Участок программы упорядочения массива.

изменения счетчика цикла нужные нам выражения $N-1$ и $J+1$ (см. § 9). Поэтому мы вводим новые переменные NM1 и IP1, равные значениям этих выражений. Идентификаторы переменных мы выбираем таким образом, чтобы они указывали на их смысл. В операторе с меткой 51 введена еще одна переменная ZX, в которой запоминается промежуточное значение i -го элемента массива X .

Упражнения. Написать участки программ, выполняющие следующие операции.

1. Имеются одномерные массивы A и B , содержащие по N элементов. Считая элементы массивов координатами n -мерных векторов a и b , вычислить их скалярное произведение

$$\text{SCAL} = \sum_{i=1}^n a_i b_i.$$

2. Вычислить след, т. е. сумму диагональных элементов матрицы ENERGY размера $M \times M$.

3. Вычислить матрицу W размера $L \times N$, равную произведению матриц U (размера $L \times M$) и V (размера $M \times N$).

$$w_{ij} = \sum_{k=1}^m u_{ik} v_{kj}.$$

4. Имеются два одномерных массива X и Y длины N . Рассматривая их элементы как координаты точек (x_1, x_2, \dots, x_n) и (y_1, y_2, \dots, y_n) в n -мерном пространстве, вычислить расстояние

между этими точками:

$$\text{DISTAN} = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}.$$

5. Пусть функция $f(x)$, определенная на отрезке $[a, b]$, задана значениями $f_i = f(x_i)$ в $n+1$ равноотстоящих узлах

$$a = x_1 < x_2 < \dots < x_n < x_{n+1} = b,$$

где n — четное число. Вычислить приближенное значение интеграла от функции $f(x)$ по формуле Симпсона:

$$\int_a^b f(x) dx \approx \frac{b-a}{3n} [f_1 + f_{n+1} + 4(f_2 + f_4 + \dots + f_n) + 2(f_3 + f_5 + \dots + f_{n-1})].$$

6. Решение системы алгебраических уравнений вида

$$A_i y_{i-1} - C_i y_i + B_i y_{i+1} = -F_i,$$

$$i = 1, 2, \dots, n; \quad A_1 = B_n = 0,$$

где A_i, B_i, C_i, F_i — известные параметры, а y_i — искомые величины, можно искать методом прогонки. Введем два массива вспомогательных величин α_i и β_i , $i = 2, 3, \dots, n$. Положим

$$\alpha_2 = \frac{B_1}{C_1}, \quad \beta_2 = \frac{F_1}{C_1}$$

и вычислим последовательно значения $\alpha_3, \beta_3, \alpha_4, \beta_4, \dots, \alpha_n, \beta_n$ по формулам

$$\alpha_{i+1} = \frac{B_i}{C_i - A_i \alpha_i}, \quad \beta_{i+1} = \frac{A_i \beta_i + F_i}{C_i - A_i \alpha_i}.$$

Тогда

$$y_n = \frac{A_n \beta_n + F_n}{C_n - A_n \alpha_n},$$

а остальные значения y_i вычисляются последовательно по формуле

$$y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1}.$$

Написать участок программы, реализующий метод прогонки.

§ 11. Функции и подпрограммы

В предыдущих параграфах уже было показано, каким образом операторы языка фортран позволяют значительно упростить запись программы. Так с

помощью оператора DO можно несколько раз повторить участок программы, не выписывая повторно относящиеся к нему операторы. В фортране имеются и другие возможности, облегчающие работу программиста. Мы переходим к изложению способа многократного обращения к некоторой последовательности операторов при различных значениях входящих в нее параметров.

11.1. Функция-оператор. Может оказаться, что в различных местах программы нужно проводить вычисления по одной и той же формуле, хотя переменные, входящие в эту формулу, могут быть различными. Например, в программе несколько раз вычисляется функция \arccos . Среди библиотечных функций (см. § 4) этой функции нет, и поэтому приходится пользоваться функцией арктангенс:

$$\arccos x = \arctg \frac{\sqrt{1-x^2}}{x}.$$

Обращение к арктангенсу просто. Чтобы найти $\arctg x$, достаточно записать оператор

$$Y = \text{ATAN}(X)$$

Спрашивается, нельзя ли каким-либо образом определить функцию \arccos , назвав ее, к примеру, ACOS, с тем чтобы для вычисления, например, $\arccos z$ достаточно было бы записать

$$Y = \text{ACOS}(Z)$$

Оказывается, что в фортране сделать это можно, и при этом различными способами. Простейшим способом определения функции является функция-оператор.

Функция-оператор имеет вид

$$F(P_1, P_2, \dots, P_k) = E$$

где F — идентификатор, название функции; P_1, P_2, \dots, P_k — идентификаторы, аргументы функции; E — выражение, т. е. последовательность операций, которую надо выполнить над аргументами, чтобы получить значение функции.

Примеры записи функций-операторов.

$$\text{ACOS}(X) = \text{ATAN}(\text{SQRT}(1 - X * X) / X)$$

$$\text{AMOD}(X, Y) = \text{SQRT}(X ** 2 + Y ** 2)$$

$$\text{DELTA}(X) = \text{SQRT}(1 - (AK * \text{SIN}(X)) ** 2)$$

По форме записи функция-оператор напоминает запись оператора присваивания, и для того, чтобы машина могла отличить их друг от друга, все функции-операторы, имеющиеся в данной программе, должны быть вынесены в самое ее начало и стоять перед первым исполняемым оператором. При счете программы вычисление значения функции-оператора производится только в случае обращения к ней из операторов программы. Обращение к функции-оператору аналогично обращению к библиотечным функциям, о которых говорилось в § 4.

Примеры. Если функции ACOS и DELTA определены как в предыдущем примере, то в программе могут быть написаны операторы вида

$$XMAX = ACOS (B/SQRT (B ** 2 - C ** 2))$$
$$TN = COS (N * ACOS (X))$$
$$ZLOG = ALOG (AK * COS (Y) + DELTA (Y))$$

Разберем подробно работу первого из них. Вычисление выражения начинается с вычисления в самых «глубоких» скобках. В данном случае машина сначала найдет в памяти значения переменных В и С, возведет их в квадрат и выполнит вычитание квадратов. Результат этого вычитания служит аргументом функции SQRT. Функция с таким идентификатором имеется среди библиотечных функций. Вычислив ее значение, машина разделит на него переменную В и возьмет результат в качестве аргумента функции ACOS. Функция с таким названием определена в программе как функция-оператор. Определение функции указывает последовательность операций, которую нужно выполнить над аргументом для того, чтобы получить значение функции. Машина выполнит все эти операции и результат вычисления, т. е. значение функции ACOS присвоит переменной XMAX.

Аргументы, указываемые при определении функции-оператора, называются *формальными параметрами*. Эти параметры формальны и им не отводится место в памяти машины, поскольку при каждом обращении к функции указываются те аргументы, над значениями которых нужно производить действия. Аргументы, перечисляемые при обращении к функции, называются *фактическими параметрами*. Формальные параметры обязательно должны быть идентификаторами. Фактические параметры могут быть константами, переменными, функциями или вообще любыми выражениями (см. § 4), как это видно из предыдущего примера.

Пример. Напишем программу вычисления корня уравнения

$$x = e^{-x}$$

методом Ньютона (см. задачу 4 в упражнениях к § 9). Текст программы приведен на рис. 12. В этой программе первым

		идентификатор	
		73	80
метка	продолжение	предложение на фортране	
1	5 6 7 10 15 20 25 30 35 40 45 50		
	PROGRAM ROOT		
C			
C	РЕШЕНИЕ УРАВНЕНИЯ $x = \exp(-x)$ МЕТОДОМ НЬЮТОНА		
C			
C	ИТЕРАЦИОННАЯ СХЕМА МЕТОДА $x = x - F(x)/F_1(x)$		
C	ГДЕ $F(x) = \exp(-x) - x$, $F_1(x) = -\exp(-x) - 1$		
C			
	$F(x) = \exp(-x) - x$		
	$F_1(x) = -\exp(-x) - 1$		
	ROOT = 0.0		
10	ZROOT = ROOT - F(ROOT)/F1(ROOT)		
	IF (ABS(F(ZROOT)) - 1.E-4) 30, 20, 20		
20	ROOT = ZROOT		
	GO TO 10		
30	PRINT 40, ZROOT		
40	FORMAT (E20.4)		
	STOP		
	END		

Рис. 12. Программа решения уравнения $x = e^{-x}$ методом Ньютона.

исполняемым оператором является оператор $ROOT = 0.0$ — задание начальной точки итерационного процесса. Функции-операторы F и F_1 определяются перед ним. При обращении к F и F_1 в операторе с меткой 10 формальный параметр X заменяется на фактический параметр $ROOT$.

11.2. Функция-подпрограмма. С помощью функции-оператора можно определить некоторое выражение, многократно встречающееся в программе. Однако возможности функции-оператора ограничены. Она находит лишь одно число в результате вычисления одного выражения. Если же вычисление функции не может быть сведено к одному выражению, а представляет собой целую последовательность операторов, то

для удобного обращения к такой последовательности она может быть оформлена в виде функции-подпрограммы.

Ради краткости будем называть *блоком* последовательность операторов, слабо связанных с остальными операторами и имеющих самостоятельное значение. В частности, это могут быть операторы, вычисляющие значение некоторой функции по значениям переменных, входящих в их выражения. В общем случае работа блока операторов состоит в вычислении одних величин, так называемых входных параметров блока, по значениям других величин, выходных его параметров. Если в процессе решения задачи отдельный блок должен работать несколько раз, то удобно каким-либо образом выделить его из основной программы, с тем чтобы обращаться к нему при необходимости. Обращение к блоку должно состоять в установке значений параметров и передаче управления на его первый оператор. Кроме того, нужно запомнить место ухода на него из основной программы, чтобы после окончания работы блока вернуться именно в это место.

Такой принцип перехода с возвратом и реализован в функции-подпрограмме. *Функцией-подпрограммой* называется блок операторов, оформленный специальным образом. В названии отражены две основные черты, присущие этому блоку. Во-первых, он представляет собой функцию, т. е. вычисляет одно число — значение функции по значениям переданных ему параметров. Во-вторых, он является подпрограммой. С понятием подпрограммы мы сталкиваемся впервые и поэтому остановимся на нем подробнее.

В § 8 говорилось о том, что программа состоит из последовательности операторов, заключенных между операторами PROGRAM и END. Работа программы начинается с первого ее оператора и продолжается до тех пор, пока машина не дойдет до оператора STOP. Не надо думать, однако, что машина выполняет при этом только операции, указанные в самой программе. Как мы уже говорили, программа может обращаться к вычислению библиотечных функций. Каждая из этих функций является также небольшой

программой, точнее, подпрограммой, поскольку она играет подчиненную роль в процессе счета. Обнаружив в операторе программы идентификатор библиотечной функции, машина вычисляет ее аргумент (фактический параметр) и с этим аргументом переходит из программы на блок вычисления значения функции, запомнив при этом место возврата, т. е. место в программе, куда нужно возвратиться после окончания работы блока. Мы будем иногда вместо термина *программа* использовать термин *основная программа*, чтобы подчеркнуть, что машина в процессе счета задачи может выполнять не только операторы, стоящие между PROGRAM и END. Блоки операторов, не входящие в основную программу, но работающие под ее управлением, т. е. в нужное время вызываемые ею, называются подпрограммами.

Подпрограмма переводится на язык машинных кодов независимо от основной программы и других программ, так что ничего «не знает» ни о метках, ни об идентификаторах, которые в них использованы. Это удобно по двум причинам. Если вы пишете большую программу, то обычно бывает трудно представить себе в целом ее структуру и взаимодействие частей. Обнаружив в результате счета, что в программе имеется ошибка, вы должны будете просмотреть сотни или даже тысячи операторов, чтобы найти один неверный. Если же программа разбита на подпрограммы, вы сможете писать и отлаживать эти блоки независимо друг от друга. Только убедившись в том, что все они работают правильно, вы составите основную программу, управляющую их работой в нужном порядке. Другое удобство заключается в следующем. Поскольку подпрограммы оформляются независимо, они могут быть использованы различными программами. Программисту не нужно писать также стандартные блоки как, например, вычисление определенного интеграла методом Симпсона или вычисление функций Бесселя. Он может легко воспользоваться подпрограммами, написанными другими программистами.

В языке фортран различаются два типа подпрограмм: функции-подпрограммы и просто подпрограммы. *Функция-подпрограмма* оформляется следующим

образом. Первым в последовательности относящихся к ней операторов должен стоять оператор

FUNCTION $F(P_1, P_2, \dots, P_k)$

где F — идентификатор, имя функции; P_1, P_2, \dots, P_k — ее аргументы, формальные параметры. Последним из операторов функции-подпрограммы должен быть оператор END. То место, в котором нужно прекратить вычисление функции и вернуться в вызывающую программу, указывается оператором возврата RETURN *).

Пример. Оформим в виде функции-подпрограммы блок операторов, выбирающий меньшее из двух чисел. Результат, т. е. меньшее число, составляет значение функции, которую мы назовем SMALL. Запись операторов представлена на рис. 13. Формальные параметры A и B являются аргументами функции. Их значения не определены до тех пор, пока программа не обратится к функции SMALL, заменив формальные параметры на фактические. Если A меньше B , то значение функции SMALL равно A . Если же A больше или равно B , значением функции будет B .

метка	продолжение
1	FUNCTION SMALL(A, B)
	IF (A - B) 1, 2, 2
1	SMALL = A
	RETURN
2	SMALL = B
	RETURN
	END

Рис. 13. Оформление функции-подпрограммы.

Идентификатор функции—это ее имя. Как и любой другой идентификатор, он должен представлять собой последовательность букв и цифр, содержащую не более шести символов и начинающуюся с буквы. Если имя функции начинается с букв I, J, K, L, M, N, то функция считается целой, т. е. ее значение будет целым. В противном случае функция считается действительной. (Именно по этой причине в качестве идентификатора библиотечной функции логарифма выбрано ALOG, а не LOG.) Формальные параметры должны быть идентификаторами простых переменных, или идентификаторами массивов. Формальный параметр не может быть, например, переменной с индексом (см. подробнее § 19). Поскольку функция-подпрограмма транслируется независимо от вызываю-

*) От английского слова return — возврат.

щей программы и ничего о ней не «знает», то формальные параметры, являющиеся идентификаторами массивов, должны быть описаны внутри самой функции-подпрограммы в операторе DIMENSION. Чтобы обратиться к функции-подпрограмме, достаточно написать ее идентификатор и перечислить в скобках нужные значения аргументов в том месте программы, где требуется вычислить значение функции.

Пример. Если функция SMALL определена как в предыдущем примере, то в программе может быть написано:

$$B = X + \text{SMALL}(Y, A)$$

При выполнении этого оператора машина обратится к функции SMALL, приняв за первый ее аргумент A текущее значение переменной Y в основной программе, за второй аргумент B текущее значение переменной A в основной программе, вычислит меньшее из них, вернется в основную программу, сложит со значением X и запомнит результат в ячейке B . Обратите внимание на то, что величины A и B основной программы не имеют никакого отношения к параметрам A и B функции-подпрограммы.

11.3. Подпрограмма. Функция-подпрограмма принимает только одно число — значение этой функции. Поэтому в виде функции-подпрограммы нельзя оформить блок операторов, вычисляющий значения нескольких величин, например массива переменных. Для оформления такого рода блоков операторов служит подпрограмма. *Подпрограмма*, так же как и функция-подпрограмма, транслируется независимо от основной программы и от других подпрограмм. Первым ее оператором является оператор*)

$$\text{SUBROUTINE } S(P_1, P_2, \dots, P_k)$$

где S — идентификатор, имя подпрограммы; P_1, P_2, \dots, P_k — формальные параметры.

Список формальных параметров у подпрограммы может отсутствовать. Как и в случае функции-подпрограммы, формальные параметры представляют собой идентификаторы простых переменных либо идентификаторы массивов. Все формальные параметры, являющиеся идентификаторами массивов, должны быть описаны в операторе DIMENSION внутри подпрограммы.

*) От английского слова subroutine — подпрограмма.

В отличие от функции-подпрограммы, в результате работы подпрограммы ее идентификатору не присваивается никакого значения. Если подпрограмма производит вычисления, то переменные, значения которых определяются в подпрограмме, должны быть включены в список формальных параметров, чтобы эти значения стали доступными для других программных единиц. Обычно работу подпрограммы можно представить как вычисление одних формальных параметров по значениям других. Но встречаются подпрограммы и другого типа. Например, можно написать подпрограмму, которая при обращении к ней выдает на печатающее устройство какое-либо сообщение.

Блок операторов, относящихся к подпрограмме, должен заканчиваться оператором END. То место в подпрограмме, дойдя до которого нужно прекратить вычисления и вернуться в вызывающую программу, отмечается оператором RETURN.

Пример. На рис. 14 дан текст подпрограммы SORT, которая упорядочивает массив чисел a_i , $i = 1, 2, \dots, n$, так что после ее работы $a_i \leq a_{i+1}$.

метка	продолжение
1	SUBROUTINE SORT(A, N)
5	DIMENSION A(N)
6	NM1 = N - 1
7	DO 1B I = 1, NM1
	IP1 = I + 1
	DO 1B J = IP1, N
	IF (A(I) - A(J)) 1B, 1B, 2B
2B	ZA = A(I)
	A(I) = A(J)
	A(J) = ZA
1B	CONTINUE
	RETURN
	END

Рис. 14. Подпрограмма упорядочения массива.

Обратите внимание на то, что A описан как одномерный массив длины N. Не надо думать, что подпрограмма действительно резервирует для себя в памяти массив такой длины и при обращении к ней пересылает туда массив фактического параметра.

Это было бы слишком расточительно и по времени, и по памяти. На самом деле при обращении к подпрограмме ей передаются адреса фактических параметров и, в частности, адреса элементов массивов, так что при работе подпрограмма производит операции с фактическими переменными. Именно поэтому аргументы функции и подпрограммы называются формальными параметрами — их нет в памяти машины.

Обращение к подпрограмме (вызов подпрограммы) производится оператором *)

$\text{CALL } S(Q_1, Q_2, \dots, Q_k)$

где S — имя подпрограммы, Q_1, Q_2, \dots, Q_k — фактические параметры.

Фактические параметры могут быть переменными, идентификаторами массивов переменных, константами или сколь угодно сложными выражениями. Встретив в программе оператор CALL , машина переходит к выполнению подпрограммы, начиная с первого ее оператора, подставив вместо формальных параметров значения соответствующих им фактических параметров. Работа подпрограммы продолжается до тех пор, пока управление не попадает на оператор RETURN , после чего происходит возврат в вызывающую программу на оператор, следующий за оператором CALL .

Пример. Если нам нужно упорядочить массив X из 25 чисел, то с помощью описанной выше подпрограммы SORT это можно сделать следующим образом:

$\text{CALL SORT}(X, 25)$

По этому оператору машина обратится к подпрограмме SORT , взяв в качестве массива A массив X и положив N равным 25. После выполнения подпрограммы по оператору RETURN управление будет возвращено в основную программу на оператор, непосредственно следующий за оператором CALL .

11.4. Оператор COMMON. Выше уже говорилось о том, что функция-подпрограмма и подпрограмма являются замкнутыми блоками операторов. Они транслируются независимо друг от друга, а связь их с вызывающей программой устанавливается через название самой подпрограммы и через последовательность ее аргументов. Поэтому, если в одной подпрограмме встречается, например, переменная $ALFA$, то

*) От английского глагола *to call* — вызывать.

эта ALFA не имеет никакого отношения к переменной ALFA другой подпрограммы или основной программы.

Поясним принцип распределения памяти машины. Основную программу, функцию-подпрограмму или подпрограмму будем для краткости называть *программной единицей*. При трансляции программной единицы встречаются переменные двух типов: формальные параметры и внутренние переменные. Под формальные параметры транслятор вообще не отводит памяти, поскольку при работе подпрограммы вместо них будут стоять фактические параметры, память под которые отведена в вызывающей программе. Под остальные переменные транслятор отводит память внутри участка памяти самой программной единицы, заменяя в операторах про-транслированной программы, т. е. в командах, идентификаторы переменных на адреса ячеек памяти машины, в которых хранятся значения этих переменных. Отсюда ясно, что переменным с одинаковыми идентификаторами, но из различных программных единиц, при трансляции будут отведены различные ячейки в памяти машины, а это и означает их независимость.

С одной стороны, такое положение удобно, поскольку при написании подпрограммы мы не должны заботиться о том, чтобы не были использованы идентификаторы, встречающиеся в других подпрограммах. Но с другой стороны, иногда бывает нужно сделать так, чтобы некоторые переменные различных подпрограмм означали одно и то же, чтобы они были общими для этих подпрограмм. Например, если мы пишем большую программу, работающую с одними и теми же массивами, то для удобства программирования ее естественно разбить на подпрограммы. Но разбиение на подпрограммы приведет к тому, что в каждую из них мы должны через формальные параметры передавать все нужные переменные и массивы. При этом списки параметров могут стать настолько длинными, что сведут на нет преимущество разбиения на подпрограммы.

Чтобы исправить подобную ситуацию, в фортране предусмотрены *глобальные переменные*. Они отлича-

ются от описанных выше *локальных переменных* тем, что ячейки, предназначенные для их хранения, не связаны ни с какой конкретной программной единицей и составляют отдельные блоки памяти, общие блоки, доступные всем программным единицам. Для описания того, что некоторая переменная программной единицы является глобальной переменной, служит оператор COMMON*), имеющий вид

COMMON P_1, P_2, \dots, P_k

где P_1, P_2, \dots, P_k — идентификаторы глобальных переменных.

Оператор COMMON должен стоять в каждой программной единице, в которой есть глобальные переменные. Это связано опять-таки с тем, что трансляция программных единиц ведется независимо. Например, если мы напишем

COMMON ALFA, BETA

в основной программе и подпрограмме, то значения переменных ALFA и BETA в них будут одними и теми же.

Следует иметь в виду, что связь глобальных переменных в фортране устанавливается не по их идентификаторам, а по месту в общем блоке памяти. Например, предыдущий оператор указывает, что ALFA есть первая переменная в общем блоке, а BETA — вторая. Если же в другой подпрограмме мы напишем

COMMON BETA

то значение переменной BETA в этой подпрограмме, как первой переменной блока общей памяти, будет совпадать со значением переменной ALFA основной программы.

Оператор COMMON является неисполняемым оператором. Подобно оператору DIMENSION он должен стоять до первого исполняемого оператора программной единицы.

*) От английского слова common — общий.

Упражнения.

1. Напишите функцию-оператор $\text{DELTA}(a, b, c, d) = ad - bc$ и используйте ее для решения системы двух линейных алгебраических уравнений

$$a_{11}x + a_{12}y = b_1,$$

$$a_{21}x + a_{22}y = b_2$$

по правилу Крамера.

2. Пусть $a_i, i = 1, 2, \dots, 5$, — заданный массив коэффициентов. Вычислите значение полинома

$$P_4(x) = a_1 + x(a_2 + x(a_3 + x(a_4 + xa_5)))$$

с помощью функции-оператора

$$\text{TERM}(a, b, x) = a + bx.$$

3. Напишите функцию-подпрограмму вычисления значения функции

$$\text{sign } x = \begin{cases} 1 & \text{при } x > 0, \\ 0 & \text{при } x = 0, \\ -1 & \text{при } x < 0. \end{cases}$$

4. Напишите функцию-подпрограмму **AVER**, вычисляющую среднее арифметическое значение элементов одномерного массива A длины N .

5. Напишите функцию-подпрограмму **SIMPS** вычисления определенного интеграла по формуле Симпсона. Формальными параметрами функции сделайте:

A, B — пределы интегрирования;

F — массив, в котором хранятся значения функции;

N — длину массива F .

6. Пусть матрица A имеет размер $N \times N$. Напишите подпрограмму транспонирования этой матрицы.

7. Напишите подпрограмму, выполняющую перемножение матриц $C = A \cdot B$, где A, B, C — матрицы размера $N \times N$.

ЧАСТЬ II

ФОРТРАН-ДУБНА

В первой части книги были изложены основные понятия языка фортран. Эти понятия оформились уже на первых этапах разработки языка и с незначительными модификациями вошли во все его современные версии. Набор операторов, описанных в первой части, довольно полон, и зачастую этими операторами пользователь может ограничиться в простых программах. Однако для создания программ со сложной логикой программ, от которой требуется качество в виде быстрого действия и экономии памяти машины, программисту приходится обращаться к более сложным конструкциям языка. Эти конструкции, к подробному описанию которых мы переходим, отчасти зависят от устройства конкретной вычислительной машины и поэтому не являются столь универсальными как ранее введенные понятия. Фактически, различия в них и составляют различия версий языка.

Во второй части книги мы будем говорить в основном о дубненской версии языка фортран, транслятор с которой работает на вычислительной машине БЭСМ-6. Вначале мы приводим некоторые сведения об этой машине, нужные для понимания особенностей версии языка, а затем переходим к последовательному описанию структуры операторов. Не следует думать, что материал второй части относится только к фортрану-Дубна. Почти все его конструкции имеются, быть может, с некоторыми отличиями, и в других версиях. В приложении 4 перечислены такие отличия от языка фортран IV для машин серии ЕС.

§ 12. Хранение информации в машине БЭСМ-6

В этом параграфе мы напомним читателю некоторые понятия, относящиеся к способу хранения информации в памяти вычислительной машины и, в частности, к способу хранения чисел.

Машинная память состоит из большого количества элементов памяти, каждый из которых имеет два устойчивых состояния. Примером элемента такого типа может служить обыкновенный тумблер: он либо включен, либо выключен. Одному из состояний приписывают значение ноль, другому значение единица и говорят, что элемент может запомнить двоичную цифру 0 или 1. Элементы памяти объединяются в *слова*, или иначе, в *ячейки памяти машины*, и каждый элемент служит одним из разрядов ячейки. Число ячеек памяти и их длина, т. е. число разрядов в одной ячейке, зависит от конструкции вычислительной машины. Например, память машины БЭСМ-6 состоит из 32 768 ячеек, каждая из которых имеет 48 разрядов.

Машина может хранить в ячейках последовательности нулей и единиц определенной длины. Поэтому для того, чтобы она могла запомнить значение некоторого числа, это число должно быть записано в виде последовательности двоичных цифр или, как говорят, представлено в двоичном коде. Мы рассмотрим два способа двоичной кодировки информации, используемые машиной БЭСМ-6.

Возьмем, например, число 1,52. Запись этого числа состоит из четырех символов: единицы, запятой, пятерки и двойки. Закодируем каждый символ комбинаций нулей и единиц, поставив в соответствие одному символу, как это сделано в БЭСМ-6, последовательность из восьми двоичных цифр. Ясно, что с помощью таких последовательностей можно закодировать всего $2^8 = 256$ символов.

Единицу закодируем как 00110001,
запятую закодируем как 00101100,
пятерку закодируем как 00110101,
а двойку закодируем как 00110010

(см. приложение 1, внутренний код данных).

Тогда число 1,52 может быть записано и запомнено в ячейке в виде двоичной последовательности

$$\underbrace{00110001}_{\text{единица}} \quad \underbrace{00101100}_{\text{запятая}} \quad \underbrace{00110101}_{\text{пять}} \quad \underbrace{00110010}_{\text{два}}$$

Описанный способ хранения чисел не удобен для выполнения над ними арифметических действий. Попробуйте, например, составить алгоритм получения кода суммы по кодам двух слагаемых.

Перейдем теперь к изложению другого способа представления числа в виде последовательности нулей и единиц. Напомним, что конкретные правила записи числа называются *системой счисления*. Наиболее употребительной в настоящее время является десятичная система счисления, в которой используются десять цифр: 0, 1, 2, ..., 8, 9. Роль цифры в записи числа зависит от того места, которое она занимает. Например, число 1,52 фактически служит сокращенной записью выражения

$$1 \cdot 10^0 + 5 \cdot 10^{-1} + 2 \cdot 10^{-2}$$

и может быть прочитано как «одна целая, пять десятых и две сотых». Цифра 1 указывает количество единиц в числе, поскольку она стоит на первом месте слева от запятой. Цифра 5 указывает количество десятых долей в числе, а цифра 2 количество сотых долей, так как они стоят соответственно на первом и втором месте справа от запятой. В общем случае, если через a_k обозначить десятичные цифры, число

$$\dots a_2 a_1 a_0, a_{-1} a_{-2} a_{-3} \dots$$

служит сокращенной записью выражения

$$\dots + a_2 \cdot 10^2 + a_1 \cdot 10^1 + a_0 \cdot 10^0 + \\ + a_{-1} \cdot 10^{-1} + a_{-2} \cdot 10^{-2} + a_{-3} \cdot 10^{-3} + \dots$$

Каждая цифра в зависимости от места указывает количество соответствующих степеней десятки, присутствующих в числе. Поэтому число десять называется *основанием системы*.

Ничто не мешает нам взять за основание системы счисления не десять, а какое-либо другое число. Например, возьмем за основание число восемь. Тогда в записи числа первая цифра слева от запятой бу-

дет указывать количество единиц в числе, вторая цифра количество восьмерок, третья количество степеней $8^2 = 64$ и т. д. Иначе говоря, в восьмеричной системе счисления число

$$\dots b_2 b_1 b_0, \quad b_{-1} b_{-2} b_{-3} \dots$$

является сокращенной записью выражения

$$\dots + b_2 \cdot 8^2 + b_1 \cdot 8^1 + b_0 \cdot 8^0 + \\ + b_{-1} \cdot 8^{-1} + b_{-2} \cdot 8^{-2} + b_{-3} \cdot 8^{-3} + \dots$$

Для записи числа в восьмеричной системе счисления нужны только восемь цифр: 0, 1, 2, ..., 6, 7. В самом деле, в этой системе счисления число восемь записывается как 10, так что цифра 8 не нужна. Отмечая индексом основание системы, мы можем записать $8_{10} = 10_8$. Цифры 0, 1, ..., 6, 7 называются *восьмеричными цифрами*.

Цель, к которой мы стремимся, состоит в записи числа с помощью нулей и единиц. Из предыдущего ясно, что мы получим нужный результат, если за основание системы счисления возьмем два, и всякое число, представленное выражением

$$\dots + c_2 \cdot 2^2 + c_1 \cdot 2^1 + c_0 \cdot 2^0 + \\ + c_{-1} \cdot 2^{-1} + c_{-2} \cdot 2^{-2} + c_{-3} \cdot 2^{-3} + \dots$$

будем записывать в виде

$$\dots c_2 c_1 c_0, \quad c_{-1} c_{-2} c_{-3} \dots$$

Здесь c_k есть 0 либо 1. Цифры 2 в двоичной системе счисления нет, так как число два в ней имеет запись 10, т. е. $2_{10} = 10_2$. Цифры 0 и 1 называются *двоичными цифрами*.

Приведем для справок таблицу 2 записи первых десяти целых чисел в десятичной, восьмеричной и двоичной системах счисления.

Таблица 2

Десятичная запись	Восьмеричная запись	Двоичная запись	Десятичная запись	Восьмеричная запись	Двоичная запись
1	1	1	6	6	110
2	2	10	7	7	111
3	3	11	8	10	1000
4	4	100	9	11	1001
5	5	101	10	12	1010

Записи числа в двоичной и восьмеричной системах счисления легко переводить друг в друга. Поскольку $8 = 2^3$, то три порядка двоичного числа отвечают одному порядку восьмеричного числа. Поэтому при переводе восьмеричного числа в двоичное нужно каждую восьмеричную цифру заменить на три двоичные цифры, имеющие то же значение, т. е. 0 заменить на 000, 1 заменить на 001, 2 заменить на 010, ..., 7 заменить на 111. При переводе двоичного числа в восьмеричное следует сгруппировать двоичные цифры по три, начиная от запятой, и заменить каждую тройку соответствующей восьмеричной цифрой, т. е. 000 заменить на 0, 001 заменить на 1, 010 заменить на 2, ..., 111 заменить на 7. Например, нетрудно проверить, что $64_8 = 110100_2$, а $1010111_2 = 127_8$.

В вычислительной машине хранение и действия над числами производятся в двоичной системе счисления. Машина БЭСМ-6 запоминает двоичное число в одной ячейки памяти, т. е. в 48 разрядах. Чтобы в ячейке могли уместиться числа широкого диапазона значений, каждое число A представляется в виде

$$A = m \cdot 2^p$$

где $|m| < 1$. Число m называется *мантиссой*, а p — двоичным порядком числа. Например, чтобы выделить мантиссу и порядок числа $5_{10} = 5_8 = 101_2$, это число в двоичной системе счисления следует записать как $0,101 \cdot 10^{11}$ (10 — это два). Мантисса числа пять в этой записи равна 101, а порядок равен 11. Значения мантиссы и порядка числа определены неоднозначно. То же число пять можно записать в виде $0,0101 \cdot 10^{100}$, сдвинув на разряд мантиссу и увеличив на единицу порядок. Поэтому среди всевозможных записей мантиссы и порядка выделяют *нормализованную запись*, при которой мантисса начинается с цифры 1. Очевидно, в нормализованной записи числа мантисса m удовлетворяет неравенству $\frac{1}{2} \leq |m| < 1$.

Для хранения мантиссы числа в ячейке машины БЭСМ-6 отводятся 40 разрядов (с 1 по 40), а для хранения порядка 6 разрядов (с 42 по 47). Разряды 48 и 41 указывают соответственно знаки порядка и числа. Шесть разрядов, отводимые под по-

	Порядок		Мантисса
48	47, , 42	41,	40, , 1

рядок, позволяют хранить значения от $000\ 000_2 = 0_{10}$ до $111\ 111_2 = 77_8 = 63_{10}$, что с учетом знака дает следующий диапазон изменения двоичного порядка: от -63_{10} до $+63_{10}$. Если перейти к десятичному порядку, то получим, что диапазон представления чисел на машине БЭСМ-6 лежит от 10^{-19} до 10^{+19} . Числа, имеющие абсолютное значение, меньше, чем 10^{-19} , считаются равными нулю. Для того чтобы числовое представление нуля в машине совпадало с логическим, т. е. выражалось нулями во всех 48 разрядах ячейки, порядок, хранящийся в машине, сдвинут относительно истинного порядка числа на 64_{10} , так что машинный порядок меняется не от -63_{10} до $+63_{10}$, а от 0 до 127_{10} . Нулевому порядку числа отвечает комбинация $1\ 000\ 000 = 100_8 = 64_{10}$ в разрядах 48 — 42. Для максимального порядка числа эта комбинация имеет вид $1\ 111\ 111 = 177_8 = 127_{10}$, а для минимального $0\ 000\ 000 = 000_8$. Правило построения машинного порядка числа из истинного его порядка можно выразить формально с помощью понятий прямого и дополнительного кода. *Прямой код* совпадает с двоичным представлением числа. *Дополнительный код* получается из двоичного представления заменой всех единиц нулями, всех нулей единицами и прибавлением единицы к младшему разряду числа. Например, число $000\ 001 = 01_8$ в дополнительном коде записывается как $111\ 110 + 1 = 111\ 111 = 77_8$. Теперь мы можем сказать, что положительный порядок числа хранится в прямом коде с единицей в разряде знака порядка, а отрицательный порядок представляется в дополнительном коде с нулем в знаке порядка.

Мантисса числа, как уже было сказано, в ячейке машины БЭСМ-6 хранится в разрядах с 1 по 40. Разряд 41-й отводится под знак числа. Положительная мантисса представляется в прямом коде с нулем в разряде знака числа. Отрицательная мантисса пред-

ставляется в дополнительном коде с единицей в разряде знака числа. Поэтому положительное число нормализовано, если в 40-м разряде стоит единица. Отрицательное число нормализовано, если в 40-м разряде стоит нуль, Машина может производить действия как над нормализованными, так и над ненормализованными числами.

Разберем способ хранения чисел в ячейке машины БЭСМ-6 на примере числа 100. Его восьмеричное представление $100 = 8^2 + 4 \cdot 8 + 4 = 144_8$, а двоичное 1100100. Мантисса числа равна 1100100, порядок равен $7_8 = 111$, знак порядка плюс, знак числа плюс. Согласно сказанному, в нормализованном виде число 100 хранится в ячейке как

$$\underbrace{1000111011001000}_{\text{порядок}} \dots 00_{\text{мантисса}}$$

Двоичная запись содержимого ячейки содержит очень много цифр и поэтому неудобна. Чтобы сделать запись более компактной, разряды ячейки объединяют по три и значения трех двоичных разрядов заменяют одной восьмеричной цифрой. В нашем примере таким образом получаем следующую запись содержимого ячейки:

$$4354 \ 4000 \ 0000 \ 0000$$

Порядок числа указывает положение запятой в мантиссе, т. е. количество старших разрядов мантиссы, отводимых под целую часть числа. В выписанном выше примере целая часть числа занимает семь разрядов мантиссы. Мы можем отвести под целую часть восемь разрядов, если сдвинем мантиссу на разряд вправо и увеличим порядок на единицу. В результате получили

$$\underbrace{10010000001100100}_{\text{порядок}} \dots 00_{\text{мантисса}}$$

При сдвиге мантиссы запись числа перестала быть нормализованной. Если нам известно, что значение числа целое, мы можем отвести под целую часть все 40 разрядов мантиссы, сдвинув ее вправо так, чтобы запятая оказалась справа от первого разряда. При этом порядок числа станет равным $40 = 50_8$, и то же

число 100 будет храниться как

1101000000 ... 01100100
порядок мантисса

или, в восьмеричной записи,

6400 0000 0000 0144

Такой способ фиксации положения запятой перед первым разрядом мантиссы используется для хранения целых величин (см. § 13).

§ 13. Константы

Работа электронной вычислительной машины заключается в обработке информации, согласно заданной машине последовательности инструкций. В языке фортран обращение к информации, хранящейся в памяти машины, может быть непосредственным или через название. Например, в предложении

TEN = FIVE + 5.0

мы обращаемся к значению переменной FIVE через ее название. Число пять мы указываем непосредственно.

Единица информации, обращение к которой происходит непосредственно, называется *константой*. В зависимости от вида информации в фортране различают семь типов констант:

- 1) целые,
- 2) вещественные,
- 3) с двойной точностью,
- 4) комплексные,
- 5) восьмеричные,
- 6) логические,
- 7) холлеритовские (текстовые).

13.1. Целые константы. Целые константы используются в фортране для изображения целых чисел. Целая константа представляет собой последовательность десятичных цифр, которой может предшествовать знак числа. Если константа не имеет знака, она считается положительной. Запись целой константы не содержит десятичную точку. В фортране-Дубна целая константа может состоять не более чем из

двенадцати десятичных цифр, т. е. ее абсолютное значение должно находиться в диапазоне от $2^{40}-1$ до 0.

Примеры записи целых констант были приведены в § 2.

В машине БЭСМ-6 целая константа хранится в полной ячейке в виде 48-разрядного ненормализованного двоичного числа (см. § 12).

Примеры восьмеричной записи содержимого ячейки.

1	хранится как	6400 0000 0000 0001
2	хранится как	6400 0000 0000 0002
—1	хранится как	6437 7777 7777 7777
—2	хранится как	6437 7777 7777 7776
100	хранится как	6400 0000 0000 0144
—100	хранится как	6437 7777 7777 7634

13.2. Вещественные константы. Вещественные константы используются для изображения действительных чисел (как целых, так и дробных). При записи вещественная константа имеет одну из следующих форм:

$$\begin{array}{ll}
 n \cdot m & n \cdot mEs \\
 n \cdot & n \cdot Es \\
 \cdot m & \cdot mEs \\
 & n \quad Es
 \end{array}$$

где n и m — последовательности десятичных цифр, указывающие целую и дробную части числа, E — символ, служащий разделителем между дробной частью числа и его порядком, а s — целая константа, указывающая десятичный порядок числа. Вещественная константа может иметь знак числа, при отсутствии которого она считается положительной. Примеры записи вещественных констант даны в § 2.

Для машины БЭСМ-6 абсолютная величина вещественной константы должна находиться в диапазоне от 10^{-19} до 10^{+19} или равняться нулю. Мантисса вещественной константы может иметь до 12 десятичных цифр.

Значение вещественной константы хранится в одной ячейке памяти машины в виде нормализованного двоичного числа, как описано в § 12.

Примеры.

1.	хранится как	4050 0000 0000 0000
—1.	хранится как	4020 0000 0000 0000
0.5	хранится как	4010 0000 0000 0000
—0.5	хранится как	3760 0000 0000 0000
100.	хранится как	4354 4000 0000 0000
—100.	хранится как	4363 4000 0000 0000

13.3. Константы с двойной точностью. Константы с двойной точностью также используются для изображения действительных чисел. В отличие от вещественных констант, их мантиссы могут содержать до 24 десятичных цифр. Запись константы с двойной точностью имеет одну из следующих форм:

$$\begin{array}{cc} n \cdot mDs & \cdot mDs \\ n \cdot Ds & n \ Ds \end{array}$$

где n и m — последовательности десятичных цифр, определяющие целую и дробную части числа, s — целая константа, обозначающая десятичный порядок числа. Символ D , отделяющий дробную часть константы от ее порядка, служит признаком константы с двойной точностью. Перед целой частью может стоять знак числа, при отсутствии которого константа считается положительной.

В фортране-Дубна абсолютная величина констант с двойной точностью должна находиться в диапазоне от 10^{1232} до 10^{-1232} или равняться нулю.

Примеры записи констант с двойной точностью.

2718.281828459045D—03
—3140D1

Константа с двойной точностью представляется в памяти машины в виде 96-разрядного двоичного числа и хранится в двух последовательных ячейках.

Роли разрядов двух ячеек распределяются при этом следующим образом:

разряд 96-й — знак порядка числа (1 для положительного),

разряды 95—90 — младшие шесть разрядов порядка,

разряд 89-й — знак числа,

разряды 88—49 — первая половина мантиссы (старшие разряды),

разряды 48—42 — старшие семь разрядов порядка,

разряд 41-й — нуль,

разряды 40—1 — вторая половина мантиссы (младшие разряды).

Благодаря такому распределению роли разрядов, константа с двойной точностью, не превосходящая установленных пределов вещественных констант, целиком хранится в первой ячейке отводимой для нее памяти в представлении, совпадающем с представлением вещественной константы. При этом вторая ячейка содержит одни нули.

Пример. Число 1. 0D0 — единица с двойной точностью — хранится как

4050 0000 0000 0000 0000 0000 0000 0000

Значение числа не выходит за границы, установленные для вещественной константы, и поэтому оно умещается целиком в первой ячейке. Во второй ячейке хранятся одни нули, а содержимое первой ячейки совпадает с кодом вещественной единицы, как это видно из сравнения с примером п. 13.2.

Число 2.18281828459045D + 0 хранится как

4110 5663 2267 3133 0005 0277 5114 5272

Мантисса этого числа имеет более двенадцати десятичных цифр, что недопустимо для вещественной константы. Старшие разряды мантиссы хранятся в первой ячейке, а неуместившиеся там младшие разряды — во второй.

13.4 Комплексные константы. Комплексные константы используются для записи комплексных чисел. Комплексная константа представляет собой пару вещественных констант, разделенных запятой и заключенных в скобки:

$$(r, i)$$

где r — вещественная константа, действительная часть комплексного числа; i — вещественная константа, мнимая часть комплексного числа.

Примеры записи комплексных констант.

Комплексное число	Запись на фортране
$1 + 62i$	(1,62.)
$-16,25 + 0,78i$	(-1.625E+1,0.78)
1	(1,0.)
i	(0.,1.)

Подчеркнем, что r и i не могут быть ни целыми константами, ни константами с двойной точностью.

Комплексная константа хранится в двух последовательных ячейках памяти машины как два вещественных числа.

13.5. Восьмеричные константы. Восьмеричная константа представляет собой последовательность восьмеричных цифр, после которой стоит символ В, являющийся признаком восьмеричной константы. В фортране-Дубна восьмеричная константа может иметь до 16 цифр. Допускаются отрицательные константы, цифрам которых предшествует знак минус.

В дубненской версии языка фортран нет восьмеричной арифметики, т. е. операций над числами в восьмеричной системе счисления. Поэтому восьмеричные константы почти не имеют практического применения и используются обычно лишь в операторе DATA для получения нестандартного содержимого ячейки.

Примеры записи восьмеричных констант.

Правильная запись

777321В
—1234567В
6400 0000 0000 0000В

Неправильная запись

781В — цифра 8 не является
восьмеричной

Восьмеричная константа хранится в одной ячейке памяти машины в виде 48-разрядного двоичного числа. Положительная константа хранится в прямом коде. У отрицательной константы часть числа, занимающая разряды ячейки с 41 по 1, представляется в дополнительном коде, а часть, занимающая разряды с 48 по 42, представляется в прямом коде.

Примеры.

1В	хранится как	0000 0000 0000 0001
0В	хранится как	0000 0000 0000 0000
—0В	хранится как	0000 0000 0000 0000
—1В	хранится как	0037 7777 7777 7777
100В	хранится как	0000 0000 0000 0100
—100В	хранится как	0037 7777 7777 7700
1234123412341234В	хранится как	1234 1234 1234 1234
—1234123412341234В	хранится как	1203 6543 6543 6544

13.6. Логические константы. Логические константы могут принимать только два значения — «истина» и «ложь». В фортране значение «истина» записывается как .TRUE., «ложь» записывается как .FALSE..

Таким образом, константы представляют собой слова TRUE (истина) или FALSE (ложь), заключенные в точки.

В машине БЭСМ-6 логическая константа хранится в целом машинном слове, причем только первый разряд указывает на значение константы. Если в нем стоит 1, то значение константы — «истина», если 0, то значение константы — «ложь».

.TRUE. хранится как 0000 0000 0000 0001

.FALSE. хранится как 0000 0000 0000 0000

13.7. Холлеритовские *) (текстовые) константы.

Холлеритовская константа является последовательностью любых допустимых символов (см. § 1) и обычно используется для выполнения заголовков и надписей при выводе информации. Константа имеет вид

nHf либо $'f'$

где n — целая константа, указывающая число символов в текстовой константе; символ H — признак типа константы; f — последовательность символов (пробел в текстовой константе также считается символом) длины n .

В фортране-дубна текстовая константа может содержать не более 120 символов ($n \leq 120$).

Напомним, что при покомпонной набивке карт пробелу отвечает отсутствие символа в соответствующей позиции. При набивке карт на УПП пробелу соответствует символ «корытце», который мы будем использовать иногда в тексте для указания пробелов.

Примеры текстовых констант

1 6H НАЧАЛЬНЫЕ ДАННЫЕ

'НАЧАЛЬНЫЕ ДАННЫЕ'

2 HA-

'ДЕЙСТВИТЕЛЬНЫЙ КОРЕНЬ'

2 6H TOO___HIGH___ACCURACY___REQUIRED

Все n символов, следующих за буквой H или заключенные в верхние кавычки, хранятся в памяти машины в специальном внутреннем коде данных (ВКД-коде, см. приложение 1). Код ВКД является вось-

*) Холлеритовская константа (Hollerith constant) получила свое название по имени Германа Холлерита (см., например, К. Джермейн, Программирование на IBM/360, «Мир», 1971).

миразрядным, т. е. каждый символ кодируется восьмиразрядным двоичным числом. Поэтому в одной 48-разрядной ячейке машины может храниться до шести символов. Первый символ хранится в разрядах 48—41, второй — в разрядах 40—33 и т. д., шестой символ хранится в разрядах 8—1.

Под текстовую константу всегда отводится целое число ячеек. Если число символов в константе не является кратным шести, оставшееся в последней ячейке место заполняется пробелами. Подчеркнем, что кодом пробела служит не ноль, а комбинация 040.

Пример. Текстовая константа

16NMATRIX__INVERSION

занимает три ячейки памяти машины. Ее восьмеричное представление выглядит следующим образом:

2324	0524	2444	2530
1004	4516	2544	2522
2464	4517	2342	0040

Проверим, какой текст хранится, например, в третьей ячейке. Для этого двоичное содержимое ячейки

$\frac{010}{2}$	$\frac{100}{4}$	$\frac{110}{6}$	$\frac{100}{4}$	$\frac{100}{4}$	$\frac{101}{5}$	$\frac{001}{1}$	$\frac{111}{7}$
$\frac{010}{2}$	$\frac{011}{3}$	$\frac{100}{4}$	$\frac{010}{2}$	$\frac{000}{0}$	$\frac{000}{0}$	$\frac{100}{4}$	$\frac{000}{0}$

надо представить в виде шести восьмиразрядных групп

01 010 011	01 001 001	01 001 111
01 001 110	00 100 000	00 100 000

после чего получаем код символов, хранящихся в этой ячейке

123 111 117 116 040 040

По таблице кодов (см. приложение 1) находим, что в ячейке хранится

SION__

— последние символы текстовой константы, дополненные пробелами.

Упражнения.

1. Записать следующие числа на фортране:

а) как целые константы:

1; 2; 6; —24; 120; 720; —5040;

б) как вещественные константы вида *n. m*:

1; 0,5; —0,166667; —0,041667; 362,88;

- в) как вещественные константы вида $n.mEs$:
 1 ; $-0,011667$; $83,333 \cdot 10^{-5}$; $-3,6288 \cdot 10^6$; 39916800 ;
- г) как константы с двойной точностью:
 1 ; $-0,00013889$; $19,841 \cdot 10^{-6}$;
 2432902008176640000 ; -95376431640625 ;
- д) как комплексные константы:
 1 ; $-i$; $1+i$; $0,56-18,28i$; $-12,25+42,875 \cdot 10^3i$;
 $+83,333 \cdot 10^{-5}$; $-0,620350i$.
2. Определить тип следующих констант:
- | | |
|---------------|------------------|
| 3.141593 | 282475249 |
| $+1.84528E+7$ | $-81450.625D82$ |
| 4H16561 | (12.96, 7.77E05) |
3. Найти ошибки в записи:
- а) целых констант
 -6 ; $5E+20$; 96059601903904 ;
- б) вещественных констант
 $25,411$; $-0.0168E-2.5$; $53.E+20$; $E-5$;
- в) комплексных констант
 $(1,0)$; $(1.D+32, -0,8)$; $(1,2, 3.14)$;
- г) логических констант
 $TRUE$ $.FOLSE$.

§ 14. Переменные

Единица информации, обращение к которой происходит посредством указания ее идентификатора (имени), называется переменной. Напомним, что идентификатором может служить любая последовательность от одного до шести буквенно-цифровых символов, первый из которых является буквой.

14.1. Типы переменных. Аналогично типам констант в фортране различают семь типов переменных. Тип переменной указывает на вид информации, хранящейся в ячейке, адрес которой определяется идентификатором переменной.

Переменная, как и константа, может быть

- 1) целой,
- 2) вещественной,
- 3) с двойной точностью,

- 4) восьмеричной,
- 5) комплексной,
- 6) логической,
- 7) холлеритовской (текстовой).

Тип константы определяется видом ее записи. Для указания типа переменной имеются специальные *операторы описания типа*. Переменная может быть указана в операторе описания типа или не указана. Если тип переменной не указан (неописанная переменная), она считается целой или вещественной. При этом переменная считается целой, если ее идентификатор начинается с букв I, J, K, L, M, N. В остальных случаях неописанная переменная считается вещественной.

Для описания типа переменных в дубненской версии языка фортран имеется пять операторов.

Тип переменных Оператор описания типа

целые	INTEGER	⟨список⟩
вещественные	REAL	⟨список⟩
с двойной точностью	DOUBLE PRECISION	⟨список⟩
комплексные	COMPLEX	⟨список⟩
логические	LOGICAL	⟨список⟩

где ⟨список⟩ обозначает последовательность идентификаторов, разделенных запятыми.

Примеры операторов описания типа.

COMPLEX A, REAL, KASPER

Согласно этому оператору переменные A, REAL и KASPER являются комплексными. Для хранения каждой из них в машине будет отведено две ячейки памяти, в первой из которых будет храниться действительная часть переменной, а во второй — мнимая часть.

REAL MATRIX, IMAGE

Оператор указывает, что переменные MATRIX и IMAGE являются вещественными. Если бы этого оператора не было, то согласно сказанному выше эти переменные считались бы целыми.

INTEGER OMEGA, A1

В ячейках OMEGA и A1 будут храниться целые числа, а не вещественные, как это было бы при отсутствии данного оператора.

Операторы описания типа переменных являются неисполняемыми операторами. Они должны находиться

в самом начале программы и предшествовать первому исполняемому оператору.

Каждый идентификатор может быть указан только в одном операторе описания типа. Если какой-либо из идентификаторов будет описан дважды, транслятор выдаст сообщение об ошибке.

14.2 Массивы переменных. Вне зависимости от типа переменной, в фортране различают простые переменные и переменные с индексом. *Простая переменная* представляет собой одну величину. *Переменная с индексом* представляет собой одну величину из массива величин, имеющих одинаковое название.

Массив есть множество последовательно расположенных ячеек памяти, имеющих общее название. Обращение к элементу массива, производится через название (идентификатор) массива с указанием индексов, определяющих положение данного элемента в массиве. Например:

WHAT — простая переменная;

WHY (5) — переменная с индексом, пятый элемент массива WHY;

WHEN (3,2) — переменная с индексом, элемент массива-матрицы WHEN, стоящий на пересечении третьей строки и второго столбца.

Размерность массива определяется количеством индексов, необходимых для указания его элементов. В фортране-Дубна массив переменных может быть однородным, двумерным или трехмерным. Массивы большей размерности не допускаются.

Массивы переменных должны быть описаны, т. е. в начале программы должны быть указаны все идентификаторы массивов переменных, их размерность и максимальные значения каждого из индексов. Описание массива может быть произведено в одном из следующих операторов:

1) в операторе описания типа переменной (см. п. 14.1),

2) в операторе DIMENSION,

3) в операторе COMMON*) (см. п. 19.8).

Каждый из этих операторов имеет вид

⟨название оператора⟩⟨список⟩

*) От английского слова common — общий.

Примеры операторов описания массивов.

REAL INDEX (10, 2, 3)

Оператор указывает, что INDEX служит идентификатором трехмерного массива вещественных переменных. Первый индекс этого массива может принимать значения от 1 до 10, второй от 1 до 2, третий от 1 до 3.

COMPLEX SIX(10), SEVEN(4,4)

SIX — вектор комплексных переменных, имеющий десять координат и занимающий двадцать ячеек в памяти машины, по две ячейки на каждую координату; SEVEN — матрица комплексных переменных размера четыре на четыре.

DIMENSION A(7,8), INCH(150)

A — прямоугольная матрица вещественных переменных, имеющая семь строк и восемь столбцов, INCH — вектор целых переменных длины 150.

Отметим, что тип массива переменных можно указать в операторе описания типа, а его размер в операторе DIMENSION.

Например, запись

DOUBLE PRECISION X, F, W
DIMENSION X(50), F(50)

эквивалентна записи

DOUBLE PRECISION X(50), F(50), W

Оператор DIMENSION, так же как и операторы описания типа переменных, относится к числу неисполняемых операторов и в программе должен предшествовать первому исполняемому оператору. Порядок неисполняемых операторов между собой несуществен.

14.3. Расположение массива в памяти машины. Идентификатор переменной указывает программе место в памяти вычислительной машины, в котором хранится значение этой переменной. В случае простой переменной идентификатор определяет адрес ячейки, хранящей ее значение. В случае массива переменных идентификатор определяет адрес первой ячейки участка памяти, хранящего этот массив. Для отыскания значения некоторой переменной с индексом машина по идентификатору переменной находит в памяти начало массива, а затем по значениям

индексов рассчитывает положение переменной относительно начала массива.

Расположение массива в памяти машины определяется следующими правилами.

Элементы двумерного массива хранятся в памяти последовательно. Например, пусть массив ABC описан как

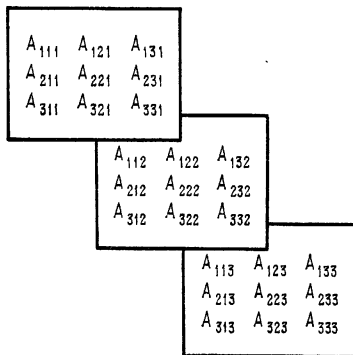
DIMENSION ABC(15)

Первой ячейке массива присваивается имя ABC. В этой ячейке хранится элемент ABC(1). Элемент ABC(2) хранится в ячейке ABC+1, элемент ABC(3) хранится в ячейке ABC+2, ..., элемент ABC(15) хранится в ячейке ABC+14.

Элементы двумерного массива хранятся в памяти машины по столбцам. Например, пусть массив AB описан как

DIMENSION AB(3,4)

Первой ячейке массива присваивается имя AB. В этой ячейке хранится элемент AB(1,1). В ячейке AB+1 хранится элемент AB(2,1), в ячейке AB+2 хранится элемент AB(3,1), в ячейке AB+3 хранится элемент AB(1,2), в ячейке AB+10 хранится элемент AB(2,4) в ячейке AB+11 хранится элемент AB(3,4).



Трехмерный массив представляет собой последовательность двумерных массивов-матриц, нумеруемых третьим индексом. Эти матрицы располагаются в памяти машины последовательно друг за другом, а каждая из них хранится по столбцам. Например, пусть трехмерный массив A описан как

Рис. 15. Расположение элементов трехмерного массива.

DIMENSION A(3, 3, 3)

Представим его в виде трех матриц, как показано на рис. 15. Первой ячейке массива в памяти машины

присваивается имя А. В этой ячейке хранится элемент А (1, 1, 1). В ячейке А + 1 хранится элемент А (2, 1, 1), в ячейке А + 2 хранится элемент А (3, 1, 1), в ячейке А + 3 хранится элемент А (1, 2, 1), ..., в ячейке А + 25 хранится элемент А (2, 3, 3), а в ячейке А + 26 хранится элемент А (3, 3, 3).

Легко установить общую формулу, указывающую номер ячейки, в которой хранится определенный элемент массива. Предположим, что этот массив описан как В(L, М, N), так что первый индекс может принимать значения от 1 до L, второй от 1 до М, третий от 1 до N. Элемент В(1, 1, 1) хранится в ячейке В. Положение элемента В(I, J, K) относительно ячейки В определяется по формуле:

$$В + (Е - 1) + (I - 1 + L * (J - 1 + M * (K - 1))) * E$$

где Е — число ячеек, отводимое для одного элемента массива. Е = 1 для целых, вещественных и логических переменных, Е = 2 для комплексных переменных и переменных с двойной точностью.

Например, для описанного выше трехмерного массива А, как указано в операторе DIMENSION, L = 3, М = 3, N = 3. Массив А состоит из вещественных переменных, так что для его элемента с индексами I, J, K предыдущая формула преобразуется к виду

$$(A + (I - 1) + 3 * (J - 1 + 3 * (K - 1))).$$

В частности, элемент А(1, 3, 3) находится в ячейке

$$A + (1 - 1) + 3 * (3 - 1 + 3 * (3 - 1)) = A + 24$$

14.4. Допустимые формы индексов. В фортране предполагается, что каждый индекс массива может принимать значения от единицы до максимального значения, указанного в описании массива. Отрицательное и нулевое значения индекса не допускаются. Если во время работы программы текущее значение индекса окажется большим, чем максимальное его значение, это может привести к ошибочным результатам счета.

В программе при обращении к элементу массива разрешается указывать меньшее число индексов, чем то, которое задается в описании массива. В этом случае для определения положения переменной

недостающие старшие индексы формально считаются равными единице.

Например, для трехмерного массива, описанного как ,

DIMENSION CAN(L, M, N)

обращение CAN(I, J, K)	подразумевает CAN(I, J, K)
обращение CAN(I, J)	подразумевает CAN(I, J, 1)
обращение CAN(I)	подразумевает CAN(I, 1, 1)
обращение CAN	подразумевает CAN(1, 1, 1)

Аналогично для двумерного массива, описанного как

DIMENSION CAT(L, M)

обращение CAT(I, J)	подразумевает CAT(I, J)
обращение CAT(I)	подразумевает CAT(I, 1)
обращение CAT	подразумевает CAT(1, 1)

Для одномерного массива, описанного как

DIMENSION CAP(L)

обращение CAP(1)	подразумевает CAP(1)
обращение CAP	подразумевает CAP(1)

Однако обращение к элементам некоторого массива, как к элементам массива большей размерности, является ошибкой. Например, если массив CAD описан как

DIMENSION CAD(21)

то обращение к его элементу CAD(3,2) не допускается.

Для указания значения индекса массива в программе может быть использована:

- 1) целая константа без знака;
- 2) простая целая переменная, т.е. переменная без индексов;
- 3) выражение вида

$$c * I \pm d, I \pm d, c * I,$$

где c и d — целые константы без знака, а I — простая целая переменная.

Другие формы индексов в фортране-Дубна не допускаются.

Примеры правильной записи индексов.

KING(2) — второй элемент массива KING;

AND(4,2) — элемент матрицы AND, стоящий на пересечении четвертой строки и второго столбца;

MASON(1,3) — элемент матрицы MASON, стоящий на пересечении первой строки и третьего столбца;

THE (3 * BETA, KAPPA + 2) — элемент матрицы THE, первый индекс которого равен выражению 3 * BETA, а второй выражению KAPPA + 2;

MASTER (10 * K276—5) — элемент одномерного массива MASTER, индекс которого равен выражению 10 * K276—5.

Примеры неправильной записи индексов.

SUMM(I + J) — в индексном выражении не допускается сложение идентификаторов;

TEMP(MIX(5)) — переменная с индексом не может служить индексом;

TIME(Y) — индекс должен быть целой переменной;

AREA (5. * M) — в индексе не допускается действительная константа.

14.5. Эквивалентность переменных. При трансляции программы, т. е. при переводе ее с фортрана на язык машинных кодов, каждой переменной, встречающейся в программе, отводится определенное место в памяти машины. При этом, естественно, разным переменным отводятся различные ячейки памяти. Однако при программировании иногда бывает удобным сделать так, чтобы два или несколько различных идентификаторов указывали на одно и то же место в памяти машины. Иными словами, иногда бывает удобным, чтобы одна переменная имела несколько названий.

Например, программа писалась двумя программистами, и один из них в своей части использовал для переменной «высота» идентификатор H, а другой идентификатор HEIGHT. Чтобы состыковать оба участка, следовало бы переписать один из них, заменив всюду H на HEIGHT или наоборот. Но можно указать машине, что переменные H и HEIGHT эквиваленты, т. е. что они означают одно и то же. В другой ситуации программисту не хватает памяти машины, но у него есть одни массивы, которые используются только на первом этапе счета, и другие массивы, которые используются только на втором этапе. Программист может разместить их в одном и том же месте памяти машины, описав эти массивы как эквивалентные.

Общий вид оператора установления эквивалентности следующий:

EQUIVALENCE (P_1, P_2, \dots), (Q_1, Q_2, \dots)...

где P_i, Q_i — идентификаторы простых переменных или переменных с одним индексом. Список P_1, P_2, \dots представляет собой группу двух или более переменных, между которыми устанавливается эквивалентность. Например, запись

EQUIVALENCE (H, HEIGHT)

означает, что имена H и HEIGHT будут присвоены одной и той же ячейке памяти, т.е. одной переменной. Оператор

EQUIVALENCE (A, B, C(10)), (D, E)

указывает, что переменные A, B и десятый элемент массива C эквивалентны между собой. Кроме того, эквивалентны переменные D и E.

Эквивалентность двух элементов массивов автоматически устанавливает эквивалентность между остальными элементами. Например, одномерные массивы A и B имеют длину четыре и пять соответственно:

DIMENSION A(4), B(5)

Оператор

EQUIVALENCE (A(3), B(2))

указывает, что третий элемент массива A находится в том же месте памяти, что и второй элемент массива B. Поскольку каждый из массивов хранится в памяти машины последовательно, расположение массивов A и B друг относительно друга будет следующим:

A(1)	
A(2)	B(1)
A(3)	B(2)
A(4)	B(3)
	B(4)
	B(5)

т.е. автоматически становятся эквивалентными следующие пары переменных:

A(2)	и	B(1)
A(4)	и	B(3)

Подчеркнем, что в операторе EQUIVALENCE переменная может содержать только один индекс. Если же нужно установить эквивалентность с элементом двух- или трехмерного массива, этот элемент следует представить как элемент одномерного массива, подсчитав соответствующее значение индекса, как указано в п. 14.3.

Например, массивы C и D, описанные как
 DIMENSION C(6), D(3,2)

располагаются в памяти машины в следующей последовательности:

Массив C: C(1), C(2), C(3), C(4), C(5), C(6)

Массив D: D(1,1), D(2,1), D(3,1), D(1,2), D(2,2), D(3,2)

Как уже говорилось выше, элемент D(1,1) есть первый элемент массива D, и к нему можно обращаться как к D(1) или просто как к D. Элемент D(2,1) есть второй элемент массива D, и к нему можно обращаться как к элементу D(2). Элемент D(3,1) есть третий элемент массива D, и к нему можно обращаться как к элементу D(3). Элемент D(1,2) есть четвертый элемент массива D, и к нему можно обращаться как к D(4), и т. д. В общем случае связь между индексами многомерного и одномерного массива находится по формуле, приведенной в п. 14.3. Если мы хотим сделать эквивалентными элементы C(4) и D(2,2), то элемент D(2,2) следует рассматривать как D(5) и записать

EQUIVALENCE (C(4), D(5))

Последний оператор установит следующее соответствие между элементами массивов C и D:

	D(1,1)
C(1)	D(2,1)
C(2)	D(3,1)
C(3)	D(1,2)
C(4)	D(2,2)
C(5)	D(3,2)
C(6)	

Пр и м е р. Если записать последовательность операторов

```
COMPLEX C
DIMENSION A(2)
EQUIVALENCE (A, C)
```


то к вещественной переменной A(1) можно обращаться как к действительной части комплексного числа C, а к вещественной переменной A(2) можно обращаться как к мнимой части комплексного числа C.

Оператор EQUIVALENCE, так же как оператор описания типа переменных и оператор DIMENSION, относится к классу неисполняемых операторов. Он должен стоять в самом начале программы до первого исполняемого оператора. Порядок неисполняемых операторов между собой несуществен.

В операторе EQUIVALENCE не допускается установление эквивалентности между элементами блоков COMMON (прямо или косвенно). Оператор EQUIVALENCE может удлинить блок COMMON, но при этом не должно меняться начало общего блока.

Более подробно последние два замечания мы разберем в п. 19.8.

14.6. Оператор DATA. Значение переменной может вычисляться в процессе решения задачи или может быть определено до начала счета. Значения переменных второго типа составляют так называемые начальные данные (initial data) программы. Одним из способов присваивания переменным начальных значений является использование оператора DATA.

Общий вид оператора DATA в фортране-Дубна следующий:

DATA(A=<список>), (B=<список>), ...

где A, B, ... — простые переменные, переменные с индексом или индентифакторы массивов; <список> представляет собой константу или последовательность констант, разделенных запятыми.

При обработке оператора DATA константы вводятся в память машины последовательно друг за другом, так что их порядок должен соответствовать порядку запоминания машиной массивов переменных (см. п. 14.3).

Примеры записи оператора DATA.

DATA (E=2.71828), (PI=3.1416)

В ячейке E будет храниться действительное число 2,71828, а в ячейке PI действительное число 3,1416.

В записи

DIMENSION EXIT(3,2)

DATA (EXIT(2,1) = 5.82E - 4)

элементу EXIT(2,1) массива EXIT будет присвоено значение $5,82 \cdot 10^{-4}$.

В случае

DIMENSION EAGLE(5)

DATA (EAGLE = 0.5, 0.75, 1.0, 1.25)

элементу EAGLE(1) присваивается значение 0,5. Элементу EAGLE(2) присваивается значение 0,75. Элементу EAGLE(3) присваивается значение 1. Элементу EAGLE(4) присваивается значение 1,25.

В записи

DIMENSION FISH(2,3)

DATA (FISH = 15., 18., 13.5, 0.8)

двумерный массив хранится в памяти машины по столбцам. В соответствии с этим приведенный выше оператор присваивает значение 15 элементу FISH(1,1), значение 18 — элементу FISH(2,1), значение 13,5 — элементу FISH(1,2) и значение 0,8 — элементу FISH(2,2).

В записи

DIMENSION NAME(3)

DATA (NAME = 17НОБРАЩЕНИЕ — МАТРИЦЫ)

в массиве NAME хранится текст ОБРАЩЕНИЕ МАТРИЦЫ. Тот же текст массиву NAME можно присвоить оператором

DATA (NAME = 'ОБРАЩЕНИЕ — МАТРИЦЫ')

Если в списке констант имеются повторения, то запись списка можно сократить, заключив повторяющиеся константы в скобки и поставив перед открывающей скобкой целую константу, равную числу повторений. Например, запись

COMPLEX Z(5)

DATA(Z = (1.,0.), (0.,1.), (0.,1.))

эквивалентна записи

COMPLEX Z(5)

DATA(Z = (1.,0.), 2((0.,1.)))

а операторы

LOGICAL MARK(4)

DATA(MARK = .TRUE., .FALSE., .TRUE., .FALSE.)

эквивалентны операторам

LOGICAL MARK(4)

DATA(MARK = 2(.TRUE., .FALSE.))

В операторе DATA допускается запись списка переменных в виде неявного цикла (см. подробнее п. 17.3) с одним ограничением: шаг изменения счетчика цикла должен равняться единице. Например, в результате работы операторов

DIMENSION TWAIN(10)

DATA((TWAIN(I), I = 1,4) = 2(+ 1.0, — 1.0))

переменным TWAIN(1) и TWAIN(3) будет присвоено значение + 1.0, а переменным TWAIN(2) и TWAIN(4) значение — 1.0.

Тип переменных в операторе DATA определяется типом присваиваемых им констант, а не оператором описания типа. Например, оператор

DIMENSION U(10)

указывает, что U — массив вещественных переменных. Однако, если мы запишем далее

DATA(U = 1.5, 8. 6HNECTOR, .TRUE.)

то в ячейке U(1) будет храниться вещественная константа 1.5, в ячейке U(2) будет храниться целая константа 8, в ячейке U(3) будет храниться текст NECTOR, а в ячейке U(4) логическая константа TRUE..

В отличие от оператора присваивания, засылка констант оператором DATA происходит во время трансляции программной единицы, а не во время ее работы. Поэтому оператор DATA является неисполняемым оператором. Он должен стоять в самом начале программы или подпрограммы до первого исполняемого оператора. Если в программной единице имеется несколько неисполняемых операторов, то порядок между ними несуществен.

В операторе DATA не допускается засылка данных в переменные, входящие в блоки COMMON. Для засылки данных в помеченные общие блоки предусмотрена специальная программная единица BLOCK DATA (см. п. 19.9).

Кроме описанного выше вида оператора DATA, в фортране-Дубна имеется другой вид этого оператора, совпадающий с оператором DATA в фортране-IV;

DATA <список переменных>/<список констант>/, ...

где <список переменных> есть последовательность простых переменных, переменных с индексом или идентификаторов массивов переменных, перечисленных через запятую; <список констант> есть последовательность констант, перечисленных через запятую.

В списке переменных допускается использование записи в виде неявного цикла (см. п. 17.3). Для указания числа повторений в списке констант вместо приведенной выше записи $k(\dots)$ используется запись k^* . Например,

DIMENSION D(5, 10)

DATA A, B, C/5.0,6.1,7.3/,D,E/25*1., 25*2.,5.1/

Оператор DATA присваивает простым переменным A, B и C начальные значения 5, 6, 1 и 7, 3 соответственно. Далее, первым 25 элементам массива D, т. е. элементам начиная с D(1,1) и кончая D(5,5) присваивается значение единица. Следующим 25 элементам массива D, начиная с D(1,6) и кончая D(5,10) присваивается значение двойка. Простой переменной E присваивается значение 5,1.

§ 15. Выражения

Выражением называют константу, переменную (простую или с индексом), функцию или комбинацию этих величин, разделенных знаками операций и скобками. Каждое выражение должно быть написано в соответствии с некоторыми правилами, изложению которых и посвящен настоящий параграф.

В фортране различают три типа выражений:

- 1) арифметическое выражение,
- 2) выражение отношения,
- 3) логическое выражение.

Результатом вычисления арифметического выражения является число. Выражения отношения и логические выражения могут быть истинными или ложными.

15.1. Арифметические выражение. Арифметические выражение указывают на выполнение арифметических операций над константами, переменными и функциями.

К арифметическим операциям относятся:

знак операции	значение операции
+	сложение,
-	вычитание,
*	умножение,
/	деление,
**	возведение в степень.

Как и в алгебре, в фортране установлено следующее старшинство арифметических операций:

** возведение в степень	класс 1
* умножение }	класс 2
/ деление }	
+ сложение }	класс 3
- вычитание }	

В выражениях и в частях выражений, не содержащих скобок, при появлении различных по классу операций вычисления производятся в порядке их старшинства. Вычисление выражений или частей выражений, не содержащих скобок и имеющих операции одного класса, производится слева направо в порядке записи операций. Выражение $A**B**C$ вычисляется как $(A**B)**C$. Если выражение содержит скобки, то его вычисление начинается с вычисления в самих внутренних скобках.

Арифметическое выражение, подобно константе и переменной, имеет тип. Тип арифметического выражения есть тип результата, получающегося при его вычислении.

В арифметическом выражении могут участвовать величины четырех типов: целые, вещественные, комплексные и заданные с двойной точностью. Величины типа восьмеричная, текстовая и логическая в арифметическом выражении не допускаются. Однако не все допустимые величины могут быть связаны зна-

ками арифметических операций. Например, нельзя складывать комплексную величину с величиной заданной с двойной точностью.

В таблицах 3 и 4 указано, какие типы констант переменных и функций могут быть объединены

Таблица 3

Результаты операций +, —, *, /

Операнды	R	I	C	DP
R	R	R	C	DP
I	R	I	C	DP
C	D	C	C	—
DP	DP	DP	—	DP

Таблица 4

Результаты операции **

Основание	Показатель			
	R	I	C	DP
R	R	R	—	DP
I	R	I	—	—
C	—	C	—	—
DP	DP	DP	—	DP

знаками арифметических операций в допустимые выражения, а также тип получаемого результата. В таблицах

R обозначает вещественную величину (REAL),

I обозначает целую величину (INTEGER),

C обозначает комплексную величину (COMPLEX),

DP обозначает величину с двойной точностью (DOUBLE PRECISION).

Знак — указывает на недопустимую комбинацию величин.

При написании целых выражений следует помнить, что результатом деления целой величины на целую является целая величина, получающаяся отбрасыванием у частного дробной части, а не его округлением. Так результатом деления 11 (целая) на 3 (целая) является 3 (целая). Поэтому выражение $I * J / K$ может дать результат, отличный от выражения $I * (J / K)$, хотя алгебраически эти выражения эквивалентны. Например, $4 * 3 / 2 = 6$, но $4 * (3 / 2) = 4$.

15.2. Выражения отношения. Выражение отношения устанавливает отношение типа *больше, равно, меньше* результатов вычисления двух арифметических выражений. Оно имеет вид

$$E_1 \langle \text{операция} \rangle E_2$$

где E_1 и E_2 — арифметические выражения, а $\langle \text{операция} \rangle$ — один из следующих знаков операции:

знак операции	значение операции
.EQ.	= (equal to) равно,
.NE.	\neq (not equal to) не равно,
.GT.	$>$ (greater than) больше,
.GE.	\geq (greater than or equal to) больше или равно,
.LT.	$<$ (less than) меньше чем,
.LE.	\leq (less than or equal to) меньше или равно.

Выражение отношения имеет значение TRUE (истина), если арифметические выражения E_1 и E_2 удовлетворяют значению операции. В противном случае выражение отношения имеет значение FALSE (ложь). E_1 и E_2 не могут быть комплексными арифметическими выражениями, поскольку для комплексных чисел не определены операции сравнения *больше* и *меньше*.

Примеры записи выражений отношения. Выражение INDEX.EQ.5 истинно, если значение переменной INDEX равно пяти. В противном случае выражение ложно.

Выражение $X + 0.5 * Y$.GT. $2.0 * Z$ истинно, если справедливо неравенство $x + y/2 > 2z$.

Выражение

$$U(I).LE.U(I+1)$$

истинно, если $u_i \leq u_{i+1}$.

15.3. Логические выражения. Логическое выражение указывает на выполнение операций над логическими переменными. К логическим операциям относятся:

знак операции	значение операции
.NOT.	логическое «не»,
.AND.	логическое «и»,
.OR.	логическое «или».

Действие логических операций определяется следующими правилами:

Логическое «не»

.NOT. TRUE.	имеет значение .FALSE.
.NOT. .FALSE.	имеет значение .TRUE.

Действия логического «и» и логического «или» сведены в таблицы 5 и 6.

Таблица 5

Значение выражения
A .AND. B

Значение A	Значение B	
	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

Таблица 6

Значение выражения
A .OR. B

Значение A	Значение B	
	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

В логическом выражении или части выражения, не содержащей скобок:

первой выполняется операция .NOT. ,

второй выполняется операция .AND. ,

последней выполняется операция .OR. .

Пример. Выражение

A .OR. .NOT. B .AND. C

следует понимать как

A .OR. ((.NOT. B) .AND. C)

Выражение

(A .OR. B) .AND. .NOT. B .OR. A .AND. B

следует понимать как

((A .OR. B) .AND. (.NOT. B)) .OR. (A .AND. B)

Операции отрицания .NOT. не могут стоять рядом. Если это требуется по смыслу выражения, то последовательность операций должна быть указана скобками:

.NOT. (.NOT. (.NOT....) ...) ...

Поскольку значение выражения отношения является логической величиной, выражение отношения может входить в качестве составной части в логическое выражение. Например, логическое выражение

X**2+Y**2.LT.1.0.AND.Y.GT.0.0

истинно, если точка (x, y) принадлежит полукругу.

В это логическое выражение входят два выражения отношения:

$$\begin{aligned} X**2+Y**2.LT.1.0 \\ Y.GT.0.0 \end{aligned}$$

в которые в свою очередь входят арифметические выражения.

Как видно из примера, в одном логическом выражении могут встречаться знаки арифметических операций, знаки операций отношения и знаки логических операций. При этом в первую очередь выполняются арифметические операции, затем операции отношения и в последнюю очередь логические операции.

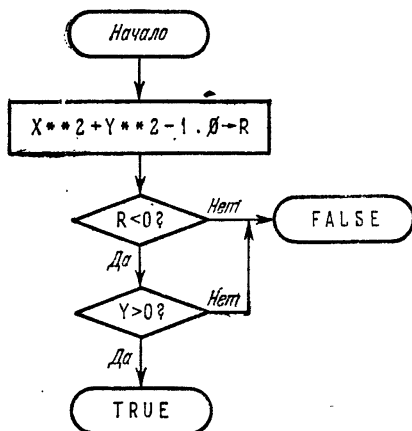


Рис. 16. Схема обработки логического выражения.

Например, обработка предыдущего выражения идет как показано на рис. 16.

§ 16. Операторы

Программа, записанная на языке фортран, представляет собой последовательность предложений или *операторов*, определяющих порядок работы вычислительной машины. О форме записи операторов на бланке мы уже говорили в § 6. К сказанному там добавим два замечания, относящиеся к дубненской версии языка.

1. Значение метки оператора должно лежать в диапазоне от 1 до 32 767.

2. На одной строке разрешается записывать несколько операторов, отделяя их друг от друга символом \$ «доллар». Например, запись последовательности операторов

$$2 \text{ ZX} = \text{X(I)}$$
$$\text{X(I)} = \text{X(J)}$$
$$\text{X(J)} = \text{ZX}$$

эквивалентна записи

$$2 \text{ ZX} = \text{X(I)} \ \$ \ \text{X(I)} = \text{X(J)} \ \$ \ \text{X(J)} = \text{ZX}$$

Операторы, следующие за знаком разделения \$, не могут иметь метку.

16.1. Исполняемые и неисполняемые операторы.

Операторы языка фортран делятся на два больших класса: на исполняемые и неисполняемые операторы. *Исполняемые операторы* производят вычисления, преобразуют информацию или передают управление в одно место программы в другое. *Неисполняемые операторы* содержат информацию о типе переменных, размещении массивов и возможностях совмещения рабочих полей в памяти машины.

К неисполняемым операторам относятся:

1) операторы описания типа переменных (см. п. 14.1)

INTEGER

REAL

DOUBLE PRECISION

COMPLEX

LOGICAL

2) операторы распределения памяти (см. пп. 14.2, 14.5, 19.8).

DIMENSION

EQUIVALENCE

COMMON

3) оператор засылки начальных значений переменных (см. п. 14.6)

DATA

Указанные выше неисполняемые операторы должны предшествовать первому исполняемому оператору в той программе или подпрограмме, в которой они появляются. Если в программе или подпрограмме имеется несколько неисполняемых операторов, то порядок их между собой безразличен.

К исполняемым операторам относятся:

- 1) операторы присваивания,
- 2) операторы передачи управления,
- 3) операторы цикла,
- 4) операторы ввода и вывода,
- 5) операторы преобразования информации.

16.2. Операторы присваивания. Оператор присваивания служит для вычисления и запоминания результата вычисления выражения. Он имеет вид

$$A = E,$$

где A — идентификатор, E — выражение. Символ присваивания $=$ означает, что переменной A присваивается значение выражения E .

Примеры записи оператора присваивания.

$$Z = (1.0, 0.0) * X + (0.0, 1.0) * Y$$

Комплексное число Z вычисляется по значению его действительной X и мнимой Y частей.

$$X(I) = X(I-1) + 5.2$$

Переменной $X(I)$ присваивается значение выражения $X(I-1) + 5.2$.

$$\text{INDEX}(5) = .\text{TRUE.}$$

Пятому элементу массива логических переменных присваивается значение «истина».

Оператор присваивания называется *арифметическим*, если E является арифметическим выражением (см. п. 15.1). В арифметическом операторе присваивания переменная A должна быть целой, вещественной, заданной с двойной точностью, либо комплексной, причем тип переменной может не совпадать с типом выражения. Например, оператор

$$\text{IMP} = 6.5$$

присваивает целой величине IMP вещественное выражение 6.5.

В случае несовпадения типов переменной и выражения арифметический оператор присваивания преоб-

разует тип выражения согласно приведенной ниже таблице 7. Прочерк означает несовместимость типов выражения и переменной.

Таблица 7

Переменная	Тип выражения			
	R	I	C	DP
R	R	R	R	R
I	I	I	I	I
C	C	<u>DP</u>	C	<u>DP</u>
DP	DP	DP	—	DP

В соответствии с таблицей преобразование типов выражений происходит следующим образом.

Оператор вида $R = I$ преобразует целую величину в величину вещественную.

Оператор вида $R = DP$ преобразует величину, заданную с двойной точностью, в вещественную. Естественно, такое преобразование не изменит значение выражения только в том случае, если значение величины с двойной точностью не выходит за диапазон значений вещественных величин.

Оператор вида $I = R$ выделяет целую часть у вещественной величины и преобразует ее в тип целая. Следует обратить внимание на то, что выделение целой части производится не округлением вещественной величины, а отбрасыванием у нее дробной части. Например, в результате выполнения оператора

$INT = 4.99$

целая переменная INT получает значение 4.

Оператор вида $I = DP$ эквивалентен последовательному действию операторов $R = DP$ и $I = R$.

Оператор $C = R$ преобразует вещественную величину в величину типа комплексная.

Оператор $DP = R$ преобразует вещественную величину в величину с двойной точностью.

Оператор $DP = I$ эквивалентен последовательному выполнению операторов $R = I$ и $DP = R$.

Оператор присваивания $A = E$ называется *логическим оператором присваивания*, если E есть логиче-

ское выражение или выражение отношения. В логическом операторе присваивания переменная A должна быть логического типа.

Пример записи логических операторов присваивания.

LOGICAL A, B, C

B = .TRUE.

A = .NOT.B

C = A.OR..NOT.B.AND.I.GT.3

16.3. Операторы передачи управления. Естественный процесс работы программы состоит в последовательном выполнении составляющих ее операторов. Для изменения этой последовательности и для повторения работы некоторых участков программы используются операторы передачи управления. Управление может быть передано только на исполняемый оператор.

Оператор безусловной передачи управления имеет вид

GO TO n

где n — метка. Этот оператор осуществляет безусловный переход к оператору с меткой n.

Пример участка программы, содержащего оператор безусловной передачи управления.

321 SIGN = 0.0

GO TO 323

322 SIGN = 1.0

323 X = X * SIGN

После выполнения оператора SIGN = 0.0 машина переходит к выполнению оператора X = X * SIGN.

В операторе безусловной передачи управления метка указывается явным образом и не может быть изменена в процессе работы программы. Аналогичный оператор, допускающий переменное значение метки, называется в фортране *оператором передачи управления по предписанию*. Он имеет вид

GO TO m, (n₁, n₂, ..., n_k).

где m — простая целая переменная, имеющая значение метки оператора, на которую нужно передать управление, а n₁, n₂, ..., n_k — список меток, возможных значений переменной m. Список меток можно опускать

и в этом случае оператор записывается как

GO TO m

Значение переменной m устанавливается не оператором присваивания, а специальным оператором ASSIGN*), имеющим вид

ASSIGN n TO m

где n — метка, значение которой приписывается простой переменной m .

Пример участка программы содержащего оператор передачи управления по предписанию.

```
A = 2.0 $ C = 1.0
ASSIGN 370 TO LABEL
GO TO 500
370 A = B
ASSIGN 380 TO LABEL
GO TO 500
380 A = B
. . . . .
500 B = C + 1.0/A
GO TO LABEL, (370, 380)
```

В зависимости от значения целой переменной LABEL после выполнения оператора с меткой 500 управление будет передано либо на оператор с меткой 370, либо на оператор с меткой 380. Подчеркнем, что оператор ASSIGN n TO m не эквивалентен оператору присваивания $m = n$, хотя и имеет аналогичные функции. В частности, после работы оператора ASSIGN 370 TO LABEL в ячейке LABEL не хранится целое число 370.

Ветвление передачи управления на языке фортран может быть организовано с помощью *вычисляемого оператора перехода*

GO TO (n_1, n_2, \dots, n_k), m

где n_i — метки операторов, на которые может быть передано управление, а m — простая целая переменная, принимающая значения от 1 до k , где k — число меток в списке.

Оператор передает управление оператору с меткой n_i , если значение переменной m равно единице,

*) От английского слова assign — приписывать.

оператору с меткой n_2 , если значение переменной m равно двойке и т. д.

Пример использования вычисляемого оператора перехода.

```
NUMBER = 2
20 GO TO(17, 18, 19), NUMBER
18 NUMBER = NUMBER + 1
GO TO 20
```

Оператор с меткой 20 выполняется два раза: после работы оператора присваивания $NUMBER = 2$ и после передачи на него управления оператором $GO TO 20$. В первом случае значение переменной $NUMBER$ равно двум и машина перейдет к выполнению оператора с меткой 18. Во втором случае значение этой переменной равно трем. Управление будет передано на оператор с меткой 19, не принадлежащий выписанному участку программы.

Перечисленные выше три оператора образуют группу операторов безусловной передачи управления. Перейдем теперь к описанию операторов условной передачи управления, которые могут сами выбирать одну из альтернатив работы программы в зависимости от того, выполняется или не выполняется некоторое условие.

Арифметический оператор условного перехода имеет вид

IF (E) n_1, n_2, n_3

где E — арифметическое выражение (не комплексное), n_1, n_2, n_3 — метки операторов.

Если значение арифметического выражения E отрицательно, происходит переход на оператор с меткой n_1 . Если значение E равно нулю, происходит переход на оператор с меткой n_2 . Если значение E положительно, происходит переход на оператор с меткой n_3 .

Пример использования арифметического оператора условного перехода.

```
IF (X) 320 321, 322
320 SIGN = -1.0 $ GO TO 323
321 SIGN = 0.0 $ GO TO 323
322 SIGN = 1.0
323 X = X * SIGN
```

Приведенная выше последовательность операторов вычисляет абсолютную величину переменной X.

Отметим, что в качестве E допускаются сколь угодно сложные арифметические выражения (см. п. 15.1), а метки n_1, n_2, n_3 не обязательно должны быть различными.

Пример. Обозначим через DELIJ переменную, принимающую значение единица при $i = j$ и нуль при $i \neq j$. Запишем участок программы ее вычисления:

```

      DELIJ = 0.0
      IF(I - J) 101, 102, 101
102  DELIJ = 1.0
101  . . . . .

```

Оператор IF передает управление на оператор с меткой 102 при $I = J$, а при $I \neq J$ на оператор с меткой 101.

Логический оператор условного перехода имеет вид

IF (L) S

где L — логическое выражение или выражение отношения, S — любой оператор, кроме оператора цикла DO или логического оператора IF.

Если выражение L истинно, то выполняется оператор S , а затем машина переходит к выполнению следующего оператора. Если выражение L ложно, оператор S не выполняется, а сразу управление передается на оператор, следующий за оператором IF.

Примеры использования логического оператора условного перехода:

```

      IF(X.LT.0.)X = -X
      XSQRT = SQRT(X)

```

— вычисляется квадратный корень из абсолютной величины X .

```

      LOGICAL A, B, C

```

```

      IF(A.AND..NOT.B.OR.C)GO TO 51

```

— если выражение, стоящее в скобках истинно, происходит переход к оператору с меткой 51. Если выражение ложно, выполняется следующий оператор.

16.4. Операторы цикла. *Оператор цикла* позволяет несколько раз повторить некоторую последовательность операторов при различных значениях некоторой целой переменной, которую называют счетчиком цикла. Оператор имеет вид

DO n $i = m_1, m_2, m_3$

где n — метка последнего оператора, принадлежащего циклу, i — простая целая переменная — счетчик цикла, m_1, m_2, m_3 — простые целые переменные или целые константы без знака: m_1 — начальное значение счетчика цикла, m_2 — конечное значение счетчика цикла,

m_3 — шаг, с которым меняется значение счетчика цикла.

Например, если мы на-пишем:

SUMM = 0.0

DO 15 K = 1, N, 1

15 SUMM = SUMM + F(K)

то в результате работы этого участка программы в ячейке SUMM накопится сумма элементов массива F от первого до N-го включительно.

Оператор DO, оператор с меткой n и все операторы, находящиеся между ними, составляют цикл DO. Операторы данного цикла выполняются сначала при значении целой переменной i , равном m_1 , затем при i , равном $m_1 + m_3$, затем при i , равном $m_1 + 2m_3$, и т. д.,

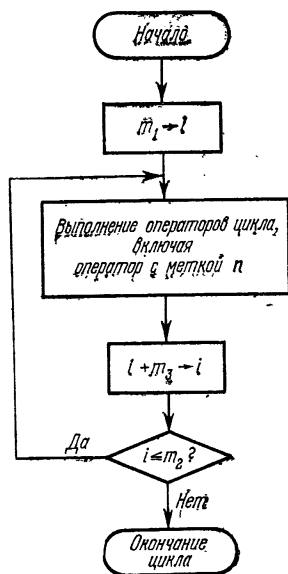


Рис. 17. Схема работы оператора цикла.

пока $i \leq m_2$. Как только значение счетчика цикла i станет больше, чем m_2 , выполнение цикла прекращается и программа переходит к выполнению оператора, следующего непосредственно за оператором с меткой n — последним оператором цикла.

Схема работы оператора цикла показана на рис. 17.

Остановимся на некоторых правилах, которые необходимо выполнять для того, чтобы избежать ошибок при записи оператора цикла.

Шаг m_3 счетчика цикла может быть только целым и положительным. Если шаг m_3 равен единице, его можно опустить при записи оператора цикла, который

в том случае будет иметь вид

DO n $i = m_1, m_2$

Операторы цикла должны выполняться хотя бы один раз, т. е. значение m_1 не должно превосходить значения m_2 .

Последним оператором цикла не может быть оператор перехода GO TO или оператор IF, или другой оператор цикла DO. Это связано с тем, что после выполнения последнего оператора цикла управление может быть передано на его начало.

Примеры правильной записи оператора цикла:

1) DO 112 KAPPA = 1, 50, 2

.

2) DO 656 I = N, M

.

3) DO 21 NOM = 10, N

Примеры неправильной записи оператора цикла:

DO 35 N 35 = 0, NEND

— цикл не может начинаться с отрицательного или нулевого значения;

DO 871 I = 53, 1, -1

— не допускается отрицательное значение шага счетчика цикла;

DO 17 K = I, N - 1

— в записи оператора цикла не допускаются выражения;

DO 97 X = 0.1, 1.0, 0.1

— счетчик цикла должен быть целой переменной.

Встречаются ситуации, в которых последний оператор цикла не может иметь метки. В этих случаях цикл можно закончить *оператором продолжения*

CONTINUE

Оператор CONTINUE является пустым оператором. Он не указывает на выполнение каких-либо вычислений, но как любой оператор он может иметь метку.

Пример использования оператора продолжения.

```
SUMM = 0.0  
DO 33 K = 1, N  
IF(FK) .LT. 0.0) GO TO 33  
SUMM = SUMM + F(K)  
33 CONTINUE
```

Этот участок программы вычисляет сумму положительных элементов массива F. Цикл состоит из четырех операторов: оператора цикла, оператора условного перехода, оператора присваивания и оператора продолжения. Мы не можем закончить цикл оператором присваивания, поскольку он должен выполняться только в том случае, когда элемент массива F(K) неотрицателен.

Значение счетчика и пределы цикла устанавливаются оператором цикла DO. Поэтому не разрешается передача управления из внешнего участка программы внутрь цикла. Однако можно передать управление изнутри цикла внешнему по отношению к нему участку программы, а потом возвратиться в тот же цикл DO. В этом случае текущее значение счетчика цикла и значения его параметров могут быть использованы при работе операторов внешнего участка.

Пример организации выхода из цикла с возвратом.

Пусть нужно составить таблицу значений функции $\cos \sqrt{x}$ для x , меняющихся от -1 до $+1$ с шагом 0.2 . Если x отрицательно, значение функции остается действительным, так как при $x < 0$ $\cos \sqrt{x} = \cos(i \sqrt{|x|}) = \text{ch} \sqrt{|x|}$. Если массив значений x имеет идентификатор X, а массив значений функции идентификатор COSSQX, мы можем, например, написать следующий участок программы:

```
DO 50 I = 1, 101  
X(I) = -1.0 + (I - 1) * 0.2  
IF(X(I) .LT. 0.0) GO TO 60  
COSSQX(I) = COS(SQRT(X(I)))  
50 CONTINUE  
.  
.  
.  
60 SQABSX = SQRT(ABS(X(I)))  
61 COSSQX(I) = (EXP(SQABSX) + EXP(-SQABSX))/2.0  
62 GO TO 50
```

Операторы с метками 60, 61 и 62 не входят в цикл DO. Однако эти операторы могут работать при выполнении цикла, и входящий в их выражения индекс I является счетчиком цикла.

Значения предела цикла m_2 и шага m_3 нельзя менять в процессе выполнения цикла.

Цикл DO может содержать внутренние циклы DO. Такая конструкция называется *гнездом DO*. В гнезде DO последний оператор внутреннего цикла должен совпадать с последним оператором внешнего цикла либо появляться до него. Если обозначить через

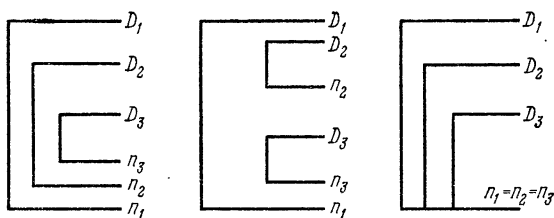


Рис. 18. Допустимые конструкции гнезд цикла.

D_1, D_2, \dots операторы цикла DO, а через n_1, n_2, \dots соответствующие им операторы окончания цикла, то это правило можно схематически изобразить так, как показано на рис. 18. Конструкция вида, показанного на рис. 19, не допускается. Максимальное число вложений циклов друг в друга равно пятидесяти.

В операторах ввода/вывода информации (см. § 17) и в операторе DATA (см. п. 14.6) при записи списка переменных можно использовать конструкцию, которая, в отличие от описанного выше оператора цикла DO, называется *неявным циклом*. Эта конструкция имеет вид

$$\langle \text{список} \rangle, i = m_1, m_2, m_3$$

где $\langle \text{список} \rangle$ есть последовательность простых переменных, переменных с индексами, массивов переменных, неявных циклов либо констант (только в операторе вывода), перечисленных через запятую; i — простая переменная, счетчик неявного цикла; m_1 — начальное значение счетчика, m_2 — конечное значение, а m_3 — шаг изменения счетчика цикла. Как и в цикле DO, параметры неявного цикла должны быть целыми константами без знака либо простыми целыми пере-

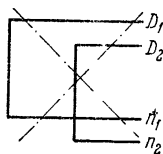


Рис. 19. Недопустимая конструкция гнезда цикла.

менными. Если шаг счетчика равен единице, его можно опускать.

Неявный цикл позволяет сократить запись списка переменных в том случае, когда построение последовательности его элементов подчиняется определенным правилам. Подобно оператору цикла DO, неявный цикл указывает, что полный список переменных получается из списка, указанного в операторе, путем перечисления его элементов вначале при значении счетчика цикла, равном m_1 , затем при значении счетчика цикла, равном $m_1 + m_3$, затем при значении $m_1 + 2m_3$ и т. д. до тех пор, пока выполняется неравенство $i \leq m_2$.

Примеры записи списка переменных с помощью неявного цикла.

Пусть Z — одномерный массив переменных. Тогда конструкция

$$(Z(I), I = 1, 5)$$

служит сокращенной записью списка

$$Z(1), Z(2), Z(3), Z(4), Z(5)$$

Конструкция

$$(Z(J), J = 1, 7, 2)$$

служит сокращенной записью списка

$$Z(1), Z(3), Z(5), Z(7)$$

Конструкция

$$(Z(K), Z(K-1), K = 2, 9, 3)$$

служит сокращенной записью списка

$$Z(2), Z(1), Z(5), Z(4), Z(8), Z(7)$$

Если ZZ — двумерный массив переменных, то неявный цикл

$$(ZZ(I, J), J = 1, 4)$$

указывает первые четыре элемента I-й строки матрицы ZZ:

$$ZZ(1, 1), ZZ(1, 2), ZZ(1, 3), ZZ(1, 4)$$

Цикл

$$(ZZ(I, J), I = 1, 4)$$

перечисляет первые четыре элемента J-го столбца матрицы ZZ:

$$ZZ(1, J), ZZ(2, J), ZZ(3, J), ZZ(4, J)$$

Цикл

$$(ZZ(K, K), K = 2, 6)$$

перечисляет диагональные элементы матрицы ZZ:

$$ZZ(2, 2), ZZ(3, 3), ZZ(4, 4), ZZ(5, 5), ZZ(6, 6)$$

Двойной цикл

$$((ZZ(L, M), M = 1, 3), L = 1, 2)$$

перечисляет элементы матрицы по строкам в порядке

ZZ(1, 1), ZZ(1, 2), ZZ(1, 3), ZZ(2, 1), ZZ(2, 2), ZZ(2, 3)

Двойной цикл

((ZZ(L, M), L = 1, 2), M = 1, 3)

перечисляет те же элементы по столбцам, т. е. в порядке

ZZ(1, 1), ZZ(2, 1), ZZ(1, 2), ZZ(2, 2), ZZ(1, 3), ZZ(2, 3)

Упражнения. Ниже приводится текст программы, отпечатанный машиной БЭСМ-6 и имеющий указания транслятора о том, что программа содержит ошибки. Найдите и объясните эти ошибки.

```

                                PROGRAM ERROR 4
                                COMPLEX AC
                                DOUBLE PRECISION ADP
                                AR = AC
2                                I = AC
3                                AC = I
4                                AC = ADP
27003
НЕЗАКОННОЕ СМЕШЕНИЕ ТИПОВ ПЕРЕМЕННЫХ В АРИФ-
                                АР = AC
05                                АР = АС
27003
НЕЗАКОННОЕ СМЕШЕНИЕ ТИПОВ ПЕРЕМЕННЫХ В АРИФ-
06                                99999 AR = 2.
                                999 BR = 21.
15032
МЕТКА И ПРИЗНАК ПРОДОЛЖЕНИЯ.
7                                ASSIGN 51 TO LABEL
8                                GO TO 311
9                                051 GO TO (99, 98, 97) I
10                               IF(AR), 96, 95, 94
24101
ОШИБКА В ТОЧКАХ РАЗВЕТВЛЕНИЯ ОПЕРАТОРА IF.
11                               99 IF(BR. GT. 15.), GO TO 96
24101
ОШИБКА В ТОЧКАХ РАЗВЕТВЛЕНИЯ ОПЕРАТОРА IF.
12                               98 IF(BR. GT. 10.)
13                               — IF(AR) 96, 95, 94
14                               97 IF(BR. GT. 5.)
15                               — IF(BR. LT. 0)
21150
ЛОГИЧЕСКИЙ IF В ЛОГИЧЕСКОМ IF ОПЕРАТОРЕ.
16                               — GO TO 96
17                               96 IF(BR. EQ. 3..AND. AR. GT. 3.)
18                               — DO 93 I = 1, 10
19                               93 I = I * I
20                               DO 92 M92 = MMAX, MMIN, -1
21006
```

ПАРАМЕТРЫ ОПЕРАТОРА DO ДОЛЖНЫ БЫТЬ ЦЕЛЫМИ
БЕЗ ЗНАКА.

21 DO 91 M91 = 0, MMAX

21006

ПАРАМЕТРЫ ОПЕРАТОРА DO ДОЛЖНЫ БЫТЬ ЦЕЛЫМИ
БЕЗ ЗНАКА.

22 91 AR = AR + BR

23 92 CONTINUE

24 SUMM = 0.0

25 DO 90 I = 1, 50

26 SUMM = SUMM + I

27 90 I = I + 1

28 DO 89 I = I, I MAX + 1

21006

ПАРАМЕТРЫ ОПЕРАТОРА DO ДОЛЖНЫ БЫТЬ ЦЕЛЫМИ
БЕЗ ЗНАКА.

29 89 SUMM = 1.
END

§ 17. Операторы ввода и вывода информации

Операторы ввода/вывода служат для обмена информацией между оперативной памятью вычислительной машины и внешними устройствами. К внешним устройствам машины БЭСМ-6 относятся магнитные барабаны, магнитные ленты, перфораторы, устройства для ввода информации с перфокарт и устройства печати на бумаге.

За каждым из подключенных к машине устройств закреплен постоянный номер. Этот номер называется *физическим номером устройства*. Физические номера работающих устройств не имеют никакого значения для программиста. Программист должен только знать, что в дубненской версии фортрана задача может использовать не более одного автоматического цифрового печатающего устройства (АЦПУ), не более 15 магнитных лент и не более одного магнитного барабана.

Программа в процессе решения задачи обращается к заказанным для нее внешним устройствам. Номера этих устройств постоянны для данной задачи и называются *математическими номерами*. Связь математических номеров с конкретными физическими номерами устанавливается лишь на время решения задачи и может меняться от запуска к запуску.

В фортране имеется своя, не совпадающая с указанной выше, нумерация внешних устройств машины.

Эти номера, используемые в операторах ввода/вывода, называются *логическими номерами* устройств. Между логическими и математическими номерами имеется строгое соответствие.

Опишем это соответствие для фортрана-Дубна. К машине БЭСМ-6 может быть подключено до 32 магнитофонов. Все магнитофоны разделены на четыре группы и каждая группа подсоединена к одному из каналов связи с машиной. Эти каналы называются третьим, четвертым, пятым и шестым направлениями. Направления первое и второе служат для связи с магнитными барабанами. В каждой группе магнитофоны имеют номера 0, 1, 2, ..., 7, а полным номером магнитофона является двузначное восьмеричное число, первая цифра которого указывает номер направления, а вторая цифра номер магнитофона в группе. Таким образом, математические номера магнитофонов имеют значения от 30₈ до 67₈.

Программа, написанная на фортране, может обращаться к магнитофонам с математическими номерами от 41₈ до 57₈. За этими магнитофонами закреплены следующие логические номера:

Логический номер	1	2	3	4	5	6	7	8
Математический номер	41	42	43	44	45	46	47	50
Логический номер	9	10	11	12	13	14	15	
Математический номер	51	52	53	54	55	56	57	

Логический номер 16 закреплен за магнитным барабаном. Логический номер 50 в операторе ввода соответствует чтению информации с перфокарт. В операторе вывода логический номер 51 соответствует печати на АЦПУ, а логический номер 52 выходному перфоратору.

Каждый оператор ввода/вывода определяет:

- 1) характер обмена информацией (чтение, печать, запись и т. д.);
- 2) логический номер устройства, с которым производится обмен;
- 3) информацию, участвующую в обмене.

Кроме того, оператор ввода/вывода может указывать, что в процессе обмена нужно произвести преобразование информации. Необходимость такого преобразования связана с тем, что в памяти машины вся информация хранится в виде двоичных чисел, а например, при выводе значений переменных на печать желательно, чтобы они имели обычный десятичный вид. Для этого при обмене информацией между оперативной памятью машины и печатающим устройством должно произойти преобразование чисел из двоичного вида в десятичный.

Описание необходимого при обмене преобразования информации содержится в операторе `FORMAT` (см. § 18). Этот оператор обязательно имеет метку, и в операторе ввода/вывода указывается метка того оператора `FORMAT`, согласно которому происходит преобразование информации. При обмене информацией в двоичном виде оператор `FORMAT` не используется.

Далее в этом параграфе используются следующие обозначения: i — логический номер внешнего устройства; он может быть либо целой константой, либо простой целой переменной; n — номер метки оператора `FORMAT`, L — список идентификаторов (подробнее в п. 17.3).

17.1. Операторы вывода. По оператору

`PRINT n, L`

машина печатает на АЦПУ текущее значение переменных, перечисленных в списке L . Информация преобразуется в строки, состоящие из 128 символов (включая пробелы), в соответствии с оператором `FORMAT`, имеющим метку n .

Печать информации на АЦПУ ведется строками, содержащими 128 символов, а строки объединяются в страницы, состоящие из 66 строк. Перед печатью очередной строки делается проверка на заполнение страницы. Если страница уже заполнена, то часть бумажной ленты прогоняется, для того чтобы информация не печаталась на сгибах, и открывается новая страница.

Программист может менять режим движения бумажной ленты. Например, можно печатать информа-

цию через строку, можно подогнать бумагу к началу новой страницы, можно не прогонять бумагу на сгибах и т. д. Для управления движением бумаги служит символ, стоящий в первой позиции строки, как показано в таблице 9. Например, если при печати строки в ее первой позиции окажется единица, то машина

Т а б л и ц а 9

Символ в первой позиции строки	Режим движения бумаги
Символ, отличный от нижеследующих	Пропуск одной строки перед печатью. Символ печатается в первой позиции
0	Пропуск двух строк перед печатью
+	Нет пропуска строки перед печатью
1	Начать новую страницу: 1 блок из 66 строк на страницу
4	Начать новый блок: 2 блока по 32 строки на страницу
5	Начать новый блок: 3 блока по 21 строке на страницу
6	Начать новый блок: 4 блока по 15 строк на страницу
S	Отмена проверки переполнения страницы

воспримет эту единицу как управляющий символ, прогонит соответствующее количество бумажной ленты и напечатает эту строку с новой страницы (уже без единицы в первой позиции). Управляющие символы 0, +, 1, 4, 5, 6, S в первой позиции не печатаются.

Оператор *)

PUNCH *n*, L

служит для выдачи на перфокарты значений переменных, указанных в списке L согласно оператору FORMAT, имеющему метку *n*.

Оператор

WRITE (*i*, *n*) L

передает информацию, определяемую списком L, устройству вывода с номером *i*. Необходимое преобразо-

*) От английского глагола to punch — пробивать (отверстие), перфорировать.

вание информации производится в соответствии с оператором FORMAT, имеющим метку n .

При значениях i от 1 до 15 происходит запись на магнитную ленту. При $i = 16$ — на магнитный барабан. При $i = 51$ оператор эквивалентен оператору PRINT. При $i = 52$ оператор эквивалентен оператору PUNCH.

Обмен информацией между оперативной памятью машины и внешними устройствами при использовании оператора FORMAT ведется определенными порциями или *физическими единицами записи*. Для АЦПУ физическая единица записи представляет собой строку из 128 символов. Для перфоратора физической единицей записи служит одна перфокарта, содержащая 80 символов (часть из которых может быть пробелами). При записи на ленту и барабан с использованием оператора FORMAT одна физическая единица записи составляет 144 символа (содержимое 24 ячеек). Физическая единица записи неделима. Нельзя, например, в одном операторе PUNCH отперфорировать число на перфокарте, а затем другим оператором PUNCH в оставшихся свободных позициях той же самой перфокарты отперфорировать еще одно число.

По оператору

WRITE(i)L

происходит запись данных, указанных списком L , на магнитную ленту (при $i = 1, 2, \dots, 15$) или магнитный барабан (при $i = 16$). Запись информации ведется в двоичных кодах, в том виде, в каком она хранится в памяти машины. Вся информация, указанная списком L , составляет одну *логическую единицу записи*. Логическая единица записи может занимать одну или несколько физических единиц записи, однако при вводе/выводе без ссылки на оператор FORMAT машина рассматривает как целое именно логическую единицу записи.

Примеры записи операторов вывода. Оператор

PRINT 10, RHO, OMEGA

указывает на печать значений переменных RHO и OMEGA под управлением оператора FORMAT с меткой 10.

Оператор

WRITE (51, 10) RHO, OMEGA

эквивалентен предыдущему.

Оператор

WRITE (NUM, 1) COPY

означает, что значение переменной COPY будет выведено с устройства, логический номер которого равен значению переменной NUM, в соответствии с оператором FORMAT, имеющим метку 1.

Оператор

WRITE (52, 11) TITLE

означает, что значение переменной TITLE будет отперфорировано на карте как описано в операторе FORMAT с меткой 11.

17.2. Операторы ввода. По оператору

READ n , L

производится ввод информации, пробитой на перфокартах, в соответствии со списком спецификаций, указанных в операторе FORMAT с меткой n . Список L определяет количество вводимой информации и ячейки в памяти машины, в которых она будет расположена.

Оператор

READ(i , n) L

вводит информацию с устройства, имеющего логический номер i , в ячейки памяти машины, определяемые списком L, в соответствии с оператором FORMAT, имеющим метку n . Значения $i = 1, 2, \dots, 15$ отвечают вводу информации с магнитных лент. При $i = 16$ происходит обмен информацией с магнитным барабаном, $i = 50$ указывает на ввод с перфокарт.

Оператор

READ(i) L

вводит информацию в двоичных кодах с устройства, имеющего логический номер i , в места памяти, определяемые списком L. Список L должен соответствовать одной логической единице записи, т. е. информация считывается с ленты или барабана теми же порциями, которыми она была записана оператором WRITE.

Примеры записи операторов ввода. Оператор

READ 110, TITLE, PH (1)

указывает на ввод с перфокарт значений переменных TITLE и PH (1) в соответствии с оператором FORMAT с меткой 110.

Оператор

READ (50, 110) TITLE, PH (1)

эквивалентен предыдущему.

Оператор

READ (L) MICKY. MAUSE

читает с устройства, номер которого равен значению переменной L, двоичные значения переменных MICKY и MAUSE.

17.3. Список оператора ввода/вывода. Оператор ввода/вывода информации указывает характер процесса (чтение/запись), внешнее устройство, с которым ведется обмен информацией, и список переменных, значения которых участвуют в обмене. Список оператора ввода/вывода определяет элементы передаваемой информации и порядок (слева направо) передачи данных. Список представляет собой последовательность элементов списка, разделенных запятыми. Например, оператор

PRINT 120, ONE, TWO, THREE

указывает на печать текущих значений переменных ONE, TWO и THREE. Последовательность переменных ONE, TWO, THREE составляет список, а каждая переменная является элементом списка. Элементы списка отделяются друг от друга запятой. Элементами списка могут быть.

- 1) простые переменные,
- 2) переменные с индексом,
- 3) массивы переменных,
- 4) неявные циклы,
- 5) константы (только при выводе).

Примеры записи списка оператора ввода/вывода.

В операторе

PRINT 12, EIN, ZWEI, DREI

элементы списка — простые переменные. Производится печать их значений как описано в операторе FORMAT с меткой 12.

В операторе

READ 27, PRES, VOL

элементы списка — простые переменные. Происходит ввод с перфокарт двух чисел. Значение первого числа присваивается переменной PRES, значение второго числа — переменной VOL.

В операторе

```
DIMENSION TRUCK (10)
```

```
.....
```

```
K=7
```

```
PRINT 777, TRUCK (2), TRUCK (K)
```

элементы списка — переменные с индексом. Они являются вторым и седьмым элементами одномерного массива TRUCK.

Индекс переменной в списке оператора ввода/вывода должен иметь одну из следующих форм (ср. п. 14.4):

$$\begin{array}{cc} c & I \\ I \pm d & c * I \\ & c * I \pm d \end{array}$$

где c и d — целые константы без знака, а I — простая целая переменная.

Если элементом списка является идентификатор массива переменных, то все элементы массива принимают участие в обмене. Например, если в программе указано:

```
DIMENSION MOWGLI(20)
```

```
.....
```

```
PRINT 291, A, MOWGLI
```

то будут отпечатаны значения простой переменной A и двадцати элементов массива MOWGLI, т. е. всего 21 число.

Ввод/вывод элементов полного массива ведется в том порядке, в котором они хранятся в памяти машины (ср. п. 14.3). Например, если CAT — матрица 2×2 :

```
DIMENSION CAT(2,2)
```

то оператор

```
PRINT 10, CAT
```

определяет следующий порядок вывода элементов:

```
CAT(1,1), CAT(2,1), CAT(1,2), CAT(2,2)
```

Для ввода/вывода части массива в качестве элемента списка можно использовать неявный цикл (см. п. 16.4). Например, если CAMEL — матрица размера 10×5 :

```
DIMENSION CAMEL (10,5)
```

то оператор

```
PRINT 8, CAMEL
```

эквивалентен оператору

```
PRINT 8, ((CAMEL(I, J), I = 1,10), J = 1,5)
```

Оба они указывают на печать массива CAMEL по столбцам. Печать по строкам осуществляется оператором

```
PRINT 8, ((CAMEL (I, J), J = 1,5), I = 1,10)
```

Наконец, при выводе элементом списка может быть константа. Например, оператор

```
PRINT 16, MONKEY, 5
```

указывает на печать двух чисел: значения переменной MONKEY и числа пять.

17.4. Управление магнитной ленты. Обмен информацией между оперативной памятью машины и магнитными лентами ведется определенными порциями — логическими единицами записи. Если чтение/запись магнитной ленты производится с указанием на оператор FORMAT, то логическая единица записи совпадает с физической единицей записи, составляющей последовательность из 144 символов (включая пробелы). Если оператор FORMAT в обмене не участвует и запись/считывание ведется двоичными кодами, то длина логической единицы записи может быть произвольной. Она равна длине участка ленты, необходимого для записи значений переменных, перечисленных в списке оператора WRITE. Например, при работе оператора

```
DIMENSION LION(25), CAT(10)
```

```
WRITE(5, 17) (LION(I), CAT(I), I = 1,10)
```

```
17 FORMAT(5(I10, F10.2))
```

будут произведены две логические единицы записи. Первая из них содержит значения переменных

```
LION (1), CAT (1), LION (7), CAT (7), ...,
```

```
LION(5), CAT(5)
```

— всего десять чисел, как указывает оператор FORMAT. Вторая запись содержит значения переменных

```
LION (6), CAT (6), LION (7), CAT (7), ....
```

```
LION(10), CAT(10)
```

Запись значений тех же переменных в двоичных кодах
WRITE(5) (LION(1), CAT(1), I = 1,10)

указывает на одну логическую единицу записи.

Считывание информации с магнитной ленты может производиться только логическими единицами записи.

Управление магнитной лентой в фортране выполняется следующими операторами.

Оператор *)

REWIND *i*

перематывает на начало магнитную ленту на *i*-м магнитофоне.

Оператор

BACKSPACE *i*

возвращает магнитную ленту на *i*-м магнитофоне на одну логическую единицу записи. Если магнитная лента была установлена на начало, оператор не исполняется. Следует учитывать, что при записи с использованием оператора FORMAT переменные списка оператора WRITE могут располагаться на нескольких логических единицах записи. В этом случае возвращение на одну логическую единицу записи, вообще говоря, не означает возврат на начало списка оператора вывода, как это происходит при обмене двоичными кодами.

Оператор

END FILE *i*

записывает признак конца участка (файла) магнитной ленты. С помощью этого оператора на ленте может быть организовано несколько участков, каждый из которых оканчивается своим признаком конца файла. Для записи информации в нужный участок ленты следует найти признак конца предыдущего файла, а потом уже производить запись. Поиск признака конца файла осуществляется в прямом направлении системной программой SCHEOF, обращение к которой имеет вид

CALL SCHEOF (*i*)

где *i* — целая константа или переменная, равная логи-

*) От английского слова to rewind — перематывать.

ческому номеру магнитной ленты. Таким образом, вызовом подпрограммы SCHEOF можно перемотать ленту вперед на один участок (файл).

Все сказанное выше об управлении магнитной лентой относится и к управлению магнитным барабаном, работа с которым ведется при $i = 16$.

17.5. Операторы ENCODE и DECODE. Операторы ENCODE и DECODE*), так же как и операторы WRITE и READ, являются операторами обмена информацией. Однако при выполнении этих операторов обмен информацией происходит не между оперативной памятью машины и внешними устройствами, а между участками оперативной памяти машины: информация передается из одного места памяти машины в другое, претерпевая при этом преобразование в соответствии со списком спецификаций оператора FORMAT.

Оператор ENCODE имеет вид

ENCODE(I, n, A)L

Здесь L — список переменных, A — идентификатор переменной или массива переменных, n — метка оператора FORMAT, I — целая константа или целая переменная, указывающая длину логической единицы записи.

Оператор ENCODE(I, n, A)L аналогичен оператору PRINTn, L. Так же как и оператор PRINT, оператор ENCODE выбирает информацию из ячеек оперативной памяти машины, адреса которых определяются списком L, преобразует ее в соответствии со списком спецификаций оператора FORMAT, имеющего метку n, а затем присваивает результат переменной (массиву переменных) A. Таким образом, по оператору ENCODE происходит преобразование информации из машинных двоичных слов в ВКД-код и упаковка ее по шести символов в ячейку. Целая константа или целая переменная I определяет длину единицы записи. Напомним, что длина единицы записи для АЦПУ равна 128 символам, для перфокарты 80 символам, для магнитной ленты 144 символам. В операторе ENCODE параметр I может определять произвольное количество символов. Если I не кратно шести, то единица записи заканчивается в середине ячейки

*) От английских слов encode — кодировать и decode — расшифровывать.

Оператор DECODE имеет вид

DECODE (I, n , A) L

где L — список переменных, A — идентификатор переменной или массива переменных, n — метка оператора FORMAT, I — длина единицы записи.

Оператор DECODE (I, n , A) L аналогичен оператору READ n , L. Информация из I последовательных символов, хранящаяся в ячейке (массиве) A преобразуется в соответствии со списком спецификаций оператора FORMAT, имеющего метку n , и запоминается в ячейках, указанных в списке L.

§ 18. Спецификации формата

Язык фортран допускает два способа хранения информации памяти машины: в виде последовательности символов и в виде двоичных машинных чисел.

Например, число 1.52 можно рассматривать как последовательность из четырех символов: 1, ., 5, 2. Каждый из этих символов можно закодировать и хранить в памяти машины получившуюся таким образом последовательность нулей и единиц. Такой способ запоминания информации удобен при вводе/выводе, поскольку он прямо указывает на то, какой из символов нужно отпечатать на бумажной ленте, отперфорировать на карте и т. д. Однако для выполнения арифметических действий над числами более естественным и удобным является не символьный способ кодировки данных, а представление их в виде двоичных машинных чисел, как это было описано в § 12. Поэтому в процессе выполнения программы числовая информация хранится в машине в двоичном представлении, а обмен информацией между оперативной памятью машины и внешними устройствами ведется символами. Исключение составляет бесформатный обмен двоичными числами с магнитными лентами и магнитным барабаном.

Необходимые для обмена конкретные виды преобразования машинных слов называются *спецификациями преобразования*. Спецификации преобразования указываются в операторе FORMAT, в соответствии с которым ведется обмен информацией.

В фортране-Дубна имеется семь спецификаций преобразования:

In — десятичное целое число,

Fn.m — десятичное вещественное число,

En.m — десятичное вещественное число с порядком,

Dn.m — десятичное число с двойной точностью,

On — восьмеричное число,

Ln — логическая величина,

Al — буквенно-цифровая информация.

Кроме указанных спецификаций преобразования, при вводе/выводе информации можно использовать:

sP — масштабный коэффициент.

Текстовые (холлеритовские) константы и управляющие редакционные спецификации:

nX — число пробелов,

/ — признак начала новой единицы записи.

18.1. Спецификация *In*. Спецификация *In* используется для ввода/вывода значений целых переменных. Здесь символ *I* — признак типа спецификации; *n* — целая константа, длина поля записи.

При выводе по спецификации *In* под число отводится поле в *n* позиций. Число размещается в правых позициях отведенного под него поля и представляется в виде

$$\underbrace{\pm \text{xx} \dots \text{x}}_n$$

где *xx ... x* — последовательность цифр. Знак \pm перед положительным числом не печатается.

Если ширина поля *n* больше, чем это требуется для записи числа, то избыточные левые позиции заполняются пробелами. Если ширина поля недостаточна для записи выводимого числа, то неуместившиеся младшие разряды числа теряются.

Пример вывода целых величин.

Пусть *I* имеет значение -3762 , *J* значение $+4762937$, а *K* значение $+13$. По оператору

PUNCH 10, I, J, K
10 FORMAT (I8, I10, I5)

перфорируется карта, представленная на рис. 20.

Под значение первой переменной отводятся первые восемь позиций на карте. Поскольку запись числа содержит пять сим-

волов, в позициях с 1-й по 3-ю пробивки нет или, как говорят иначе, эти позиции заполнены пробелами. Значение переменной J перфорируется в следующих десяти позициях, т. е. в позициях с 9-й по 18-ю. Запись числа, имеющая семь символов, располагается в позициях с 12-й по 18-ю. В позициях с 9-й по 11-ю стоят пробелы. Значение третьей переменной занимает следующие пять позиций, т. е. позиции с 19-й по 23-ю. Единица отперфорирована в 22-й позиции, а тройка в 23-й позиции.

При вводе по спецификации In поле ввода состоит из n позиций. Символы, содержащиеся в

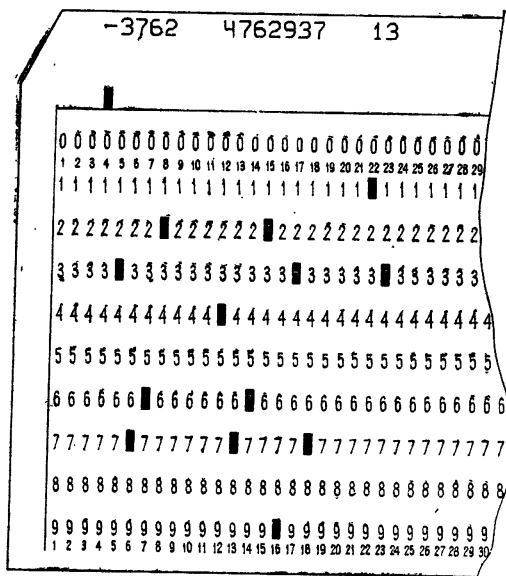


Рис. 20. Перфокарта вывода целых величин.

поле ввода, должны быть пробелами либо цифрами, которым могут предшествовать знаки + и -. Пробелы игнорируются.

Пример ввода целых величин. Пусть в программе встречаются операторы

```
READ 20, I, J, K, L, M, N
20 FORMAT (13, 17, 12, 13, 12, 14)
```

при работе которых вводится перфокарта, представленная на рис. 21. Оператор FORMAT указывает, что значение переменной I пробито на перфокарте в первых трех позициях; значение

переменной J пробито в следующих семи позициях, т. е. в позициях с 4-й по 10-ю; значение К в следующих двух позициях, в 11-й и 12-й; значение L в трех позициях с 13-й по 15-ю; значение М в 16-й и 17-й позициях, а значение переменной N пробито

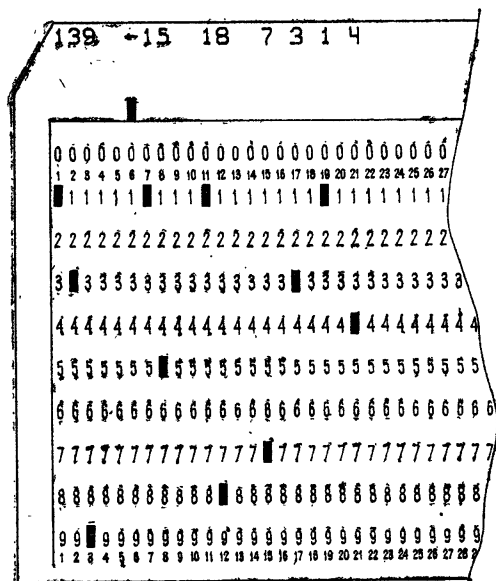


Рис. 21. Перфокарта ввода целых величин.

в следующих четырех позициях, т. е. в позициях с 18-й по 21-ю. После ввода переменная I имеет значение 139, J значение —15, К значение 18, L значение 7, М значение 3, N значение 14.

18.2 Спецификация $F_{n.m}$. Спецификация $F_{n.m}$ используется для ввода/вывода значений вещественных переменных. Здесь символ F — признак типа спецификации; n — целая константа, длина поля записи; m — целая константа, длина дробной части числа.

При выводе по спецификации $F_{n.m}$ под число отводится поле в n позиций. Число размещается в правых позициях отведенного для него поля и представляется в виде

$$\underbrace{\pm xx \dots x \cdot \overbrace{xx \dots x}^m}_{n}$$

где $xx \dots x$ — последовательность цифр. Дробная часть числа, стоящая после десятичной точки, имеет m знаков. Знак $+$ перед положительным числом не ставится.

Если поле содержит больше позиций, чем это требуется для записи числа, избыточные левые позиции заполняются пробелами. Если ширина поля недостаточна для записи выводимого числа, то в крайней левой позиции поля печатается звездочка и неуместившиеся старшие разряды числа не выводятся.

Пример. Пусть значение A равно 123,456; значение B равно -65,4321; значение C равно -87,642. Результатом работы операторов

PUNCH 30, A, B, C

30 FORMAT (F10.4, F9.3, F7.4)

является перфокарта, представленная на рис. 22.

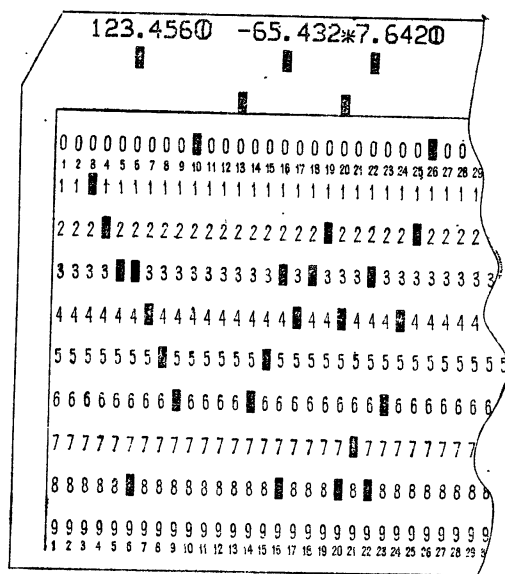


Рис. 22. Перфокарта вывода вещественных величин по F-спецификации.

Значение переменной A отперфорировано в позициях с 1-й по 10-ю. Число имеет четыре цифры после десятичной точки, так что точка стоит в шестой позиции. Следующие девять позиций с

11-й по 19-ю отводятся под значение переменной В. Переменная С перфорируется в семи позициях с 20-й по 26-ю. В поле ввода четыре позиции (с 23-й по 26-ю) отводятся под дробную часть числа, одна позиция (22-я) отводится под десятичную точку, так что под целую часть и под знак числа остается всего две позиции (20-я и 21-я), в то время как нужно отперфорировать три символа — 87. Поэтому в 21-й позиции выводится семерка, а в 20-й позиции ставится звездочка, указывающая на то, что в отведенных позициях не хватило места для вывода значений переменной С.

При вводе по спецификации $En.m$ поле ввода состоит из n позиций. В общем случае спецификации F соответствует следующая запись числа в поле ввода:

$$\pm \underbrace{\text{xx} \dots \text{x}}_{\text{целая часть}} \cdot \underbrace{\text{xx} \dots \text{x}}_{\text{дробная часть}}$$

Если знак числа отсутствует, число считается положительным. Пробелы в поле ввода игнорируются.

В частности, запись константы может содержать только целую или только дробную часть, как показано в примерах:

- 32.7216 — присутствуют целая и дробная части,
- + 1326 — только целая часть,
- .71932 — только дробная часть.

Если поле ввода не содержит десятичную точку (присутствует только целая часть числа), то в спецификации $Fn.m$ число m рассматривается как отрицательная десятичная степень масштабного множителя, т. е. как длина дробной части. После ввода в машине хранится число $\langle \text{целая часть} \rangle \times 10^{-m}$. Например, число 1357, вводимое по спецификации F4.4 будет преобразовано при вводе и запомнится как $1357 \times 10^{-4} = 0,1357$.

Если в записи вводимого числа имеется десятичная точка, значение m игнорируется. Например, число 246.753 можно вводить как по спецификации F7.3, так и по спецификации F7.0. Результат будет одним и тем же.

Указанная в спецификации длина поля ввода n должна совпадать с длиной поля, содержащего вводимое число. Несовпадение указанной и действительной длин поля может привести к ошибке.

Пример. Пусть два числа 234,2468 и 24,73314 пробиты на перфокарте как показано на рис. 23. Ввод этой перфокарты операторами

READ 8, V, T

8 FORMAT (F9.2, F9.2)

дает следующий результат: значение переменной V равно

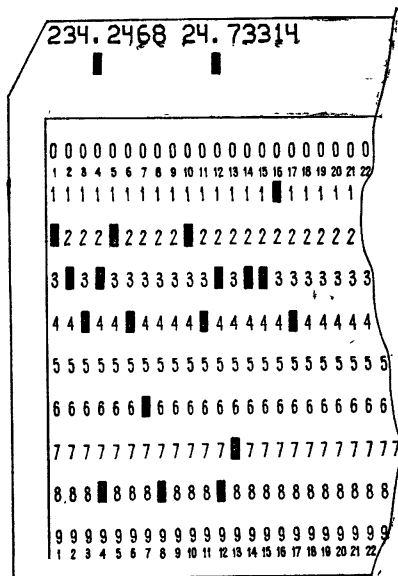


Рис. 23. Перфокарта ввода вещественных величин F-спецификации.

234.2468, а значение T равно 24.73314. Если ту же перфокарту вводить операторами

READ 9, V, T

9 FORMAT (F7.3, F8.3)

то символ 8, пробитый в восьмой позиции, будет отнесен к полю ввода переменной T. Кроме того, в это поле, занимающее позиции с 8-й по 15-ю не войдут символы 1 и 4, пробитые на перфокарте в 16-й и 17-й позициях. Поэтому в результате мы получим, что переменная V имеет значение 234.246, а значение переменной T равно 824.733.

18.3. Спецификация Еп.т. Спецификация Еп.т используется для ввода/вывода значений вещественных переменных с порядком. Здесь Е — символ, при-

знак типа спецификации; n — целая константа, длина поля записи; m — целая константа, длина дробной части числа.

При выводе по спецификации $E_{n,m}$ под число отводится поле в n позиций. Число размещается в правых позициях отведенного для него поля и представляется в виде

$$\underbrace{\pm x . \overbrace{xx \dots x}^m \pm ee}_n$$

Дробная часть, стоящая после десятичной точки, имеет m цифр. Знак $+$ перед положительным числом не ставится. Перед десятичной точкой печатается одна цифра (не ноль). Последовательность из двух цифр ее указывает десятичный порядок числа. Порядку числа обязательно предшествует знак порядка.

Если поле вывода содержит больше позиций, чем это требуется для записи числа, то избыточные левые позиции заполняются пробелами. Если ширина поля недостаточна для записи выводимого числа, то в крайней левой позиции поля печатается звездочка, а неуместившиеся старшие цифры числа не выводятся. Для того чтобы правильно указать ширину поля вывода, следует учесть, что, кроме $m + 1$ цифр числа (m цифр составляют дробную часть числа и одна цифра стоит перед десятичной точкой), в поле вывода должны уместиться:

знак числа — одна позиция,
десятичная точка — одна позиция,
знак порядка — одна позиция,
порядок числа — две позиции.

Таким образом, для правильного вывода числа по спецификации $E_{n,m}$ должно выполняться условие $n \geq m + 6$.

Пример вывода величин по E-спецификации:

DIMENSION PRINT (4)

DATA (PRINT = 0.02197E + 1, -72.1,

* 258055E - 3, -10.3861E - 2)

PRINT 20, PRINT

20 FORMAT (E15.2, E15.4, E10.5, E15.0)

Результат работы участка программы:

$$2.20 - 01 \quad \downarrow \text{15-я позиция} \quad - 7.2100 + 01 * .58055 + 02 \quad \downarrow \text{40-я позиция} \quad - 1. - 01$$

У третьего числа не хватило места для целой части.

При вводе по спецификации $E_n.m$ поле ввода состоит из n позиций. В общем случае спецификации E соответствует следующая запись числа в поле ввода:

$$\pm xx \dots x . xx \dots xE + ee$$

Пробелы в поле ввода игнорируются. Если знак числа отсутствует, число считается положительным. Десятичный порядок числа может начинаться либо с символа E , либо со знака порядка. При наличии символа E знак $+$ перед порядком можно опускать. Приведем примеры записи констант в E -спецификации:

$+1.42510E-02$	присутствуют целая, дробная части и порядок,
$- 22.003$	— целая и дробная части;
$+ 419+4$	— целая часть и порядок;
$.727E-1$	— дробная часть и порядок;
$.271828$	— только дробная часть;
2062	— только целая часть;

$E2$ (интерпретируется как нуль) присутствует только порядок.

Если поле ввода не содержит десятичную точку (присутствует только целая часть или целая часть и порядок), то в спецификации $E_n.m$ число m рассматривается как отрицательная десятичная степень масштабного множителя. При этом если d — порядок числа, то после ввода в памяти машины хранится следующее число: $\langle \text{целая часть} \rangle \cdot 10^d \cdot 10^{-m}$. Например, число $3267+05$, вводимое по спецификации $E7.8$, будет преобразовано при вводе и запомнено как $3267 \cdot 10^5 \cdot 10^{-8} = 3.267$.

Если в записи вводимого числа имеется десятичная точка, значение m игнорируется. Например, число $+3.672E5$ можно вводить как по спецификации $E8.3$, так и по спецификации $E8.0$. Результат будет одним и тем же.

Указанная в спецификации длина поля n должна совпадать с длиной поля, содержащего вводимое число. Несовпадение указанной и действительной длин поля ввода может привести к ошибке.

Пример. Пусть три числа 0.647, 2.36, 532.1 пробиты на перфокарте, как показано на рис. 24. Мы получим правильные

The diagram shows a punched card with 30 columns. The top row contains the text: +6.47E-01-2.36+5.321E+02. Below this, the card is divided into three main sections corresponding to the three numbers. Each section contains a row of digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) repeated across the columns, with markers indicating the end of each number's data field.

Рис. 24. Перфокарта ввода вещественных величин по Е-спецификации.

значения чисел, если будем вводить эту карту, например, по оператору

FORMAT (E9.2, E5.2, E10.3)

Если же эту карту вводить по оператору

FORMAT (E9.3, E7.2, E10.3)

то первым будет прочитано число $6.47 \cdot 10^{-1}$ (как и должно быть), вторым будет прочитано число $-2.36 + 5 = -2.36 \cdot 10^5$ (вместо -2.36), третьим будет прочитано число $321E+02 = 32.1$ (вместо 532.1).

18.4. Спецификация Dл.т. Спецификация Dл.т. используется для ввода/вывода значений переменных,

имеющих двойную точность. Здесь D — символ, признак типа спецификации; n — целая константа, длина поля записи; m — целая константа, длина дробной части числа.

Вывод по спецификации $Dn.m$ аналогичен выводу по спецификации $En.m$, но допускает большую длину дробной части и большую величину порядка (см. п. 13.3). При этом, однако, следует иметь в виду, что порядок числа может состоять на более чем из трех десятичных цифр.

Пример. Рассмотрим работу последовательности операторов:

```
DOUBLE PRECISION KING (3)
KING (1) = 192.89D 523
KING (2) = 2.D + 504
KING (3) = KING (1) * KING (2)
PUNCH 12, KING
12 FORMAT (3D 15.5)
```

Результат перфорации приведен на рис. 25. При выводе значения переменной KING (3) не хватило места для порядка числа.

Ввод по спецификации $Dn.m$ аналогичен вводу по спецификации $En.m$.

18.5. Спецификация On . Спецификация On используется для ввода/вывода восьмеричных целых величин. Здесь O — символ, признак типа спецификации; n — целая константа, указывающая длину поля записи.

При выводе по спецификации On под число отводится поле в n позиций. Если $n < 16$, то выводятся n младших восьмеричных цифр. Если $n \geq 16$ выводятся 16 восьмеричных цифр — содержимое одной ячейки памяти машины. Оставшиеся левые позиции поля заполняются пробелами. Отрицательное восьмеричное число представляется в дополнительном коде (см. п. 13.5).

Пример вывода по спецификации On . В результате работы последовательности операторов

```
DIMENSION U (6)
DATA U/100B, -100B, 1.5, 8, 6HNECTOR, .TRUE./
PRINT 2, U
2 FORMAT (O20)
```

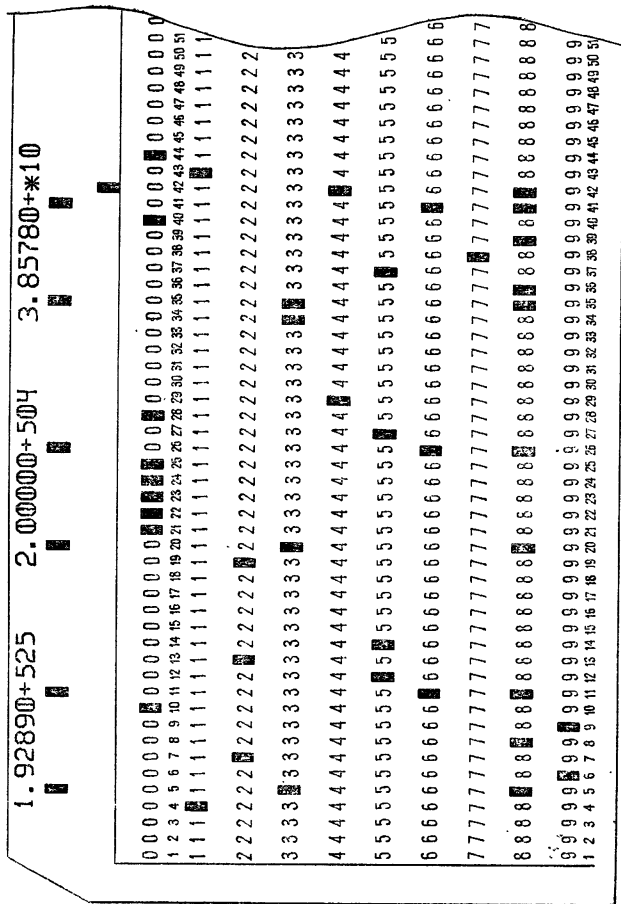


Рис. 25. Перфокарта вывода по спецификации D.

получим на выдаче:

20-я позиция
↓

```

0000000000000100
003777777777700
4053000000000000
6400000000000000
2204250325047522
0000000000000001
  
```

При вводе по спецификации *On* поле ввода состоит из *n* позиций. Вводимое число может содержать до 16 восьмеричных цифр. Допускается

Рис. 26. Перфокарта ввода по спецификации *O*.

знак числа, который должен предшествовать первой цифре. Пробелы в поле ввода игнорируются.

Пример. Рассмотрим работу операторов:

```

READ 10, P, Q, R
10 FORMAT (O10, O12, O2)
PRINT 11, P, Q, R
11 FORMAT (O20)
  
```

Вводимая перфокарта представлена на рис. 26. Результат печати выглядит следующим образом:

20-я позиция
↓
0000003737373737
0000006666644444
7777777777777777

18.6. Спецификация L_n. Спецификация L_n используется для ввода/вывода значений логических переменных. Здесь L — символ, признак типа спецификации; *n* — целая константа, длина поля записи.

При выводе по спецификации L_n поле вывода имеет длину *n* позиций. В крайней правой позиции поля выводится символ T, если значение переменной истинно (TRUE), и символ F, если значение переменной ложно (FALSE). Остальные позиции поля заполняются пробелами.

Пример. В результате работы операторов

```
LOGICAL I, J, K, L
I = .TRUE.
J = .FALSE.
K = .TRUE.
L = .TRUE.
PRINT I1, I, J, K, L
I1 FORMAT (L2, L3, L4, L5)
```

печатается строка:

_ T _ _ F _ _ _ T _ _ _ _ T

Здесь символом _ указаны пробелы.

При вводе по спецификации L_n поле ввода состоит из *n* позиций. В память машины будет занесено значение .TRUE. или .FALSE., если первый из символов поля ввода, не являющийся пробелом, есть буква T или F соответственно.

Пример. Участок программы

```
LOGICAL C (4)
READ 5, C
5 FORMAT (4 L 5)
PRINT 6, C
6 FORMAT (L 10)
```


Вводимая перфокарта изображена на рис. 27. Результат печати:

10-я позиция

↓
T
F
F
F

18.7. Спецификация Ал. Спецификация Ал используется для ввода/вывода текстовой информации.

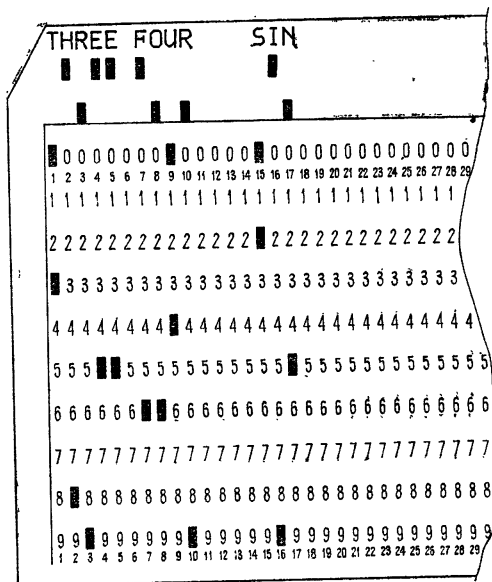


Рис. 27. Перфокарта ввода по спецификации L.

Здесь А — символ, признак типа спецификации; n — целая константа, длина поля записи.

При выводе по спецификации Ал предполагается, что содержимое ячейки представляет собой набор символов, записанных в ВКД-коде (см. приложение 1). В этом коде один символ кодируется восемью двоичными разрядами, так что в одной ячейке памяти машины БЭСМ-6, имеющей 48 разрядов, может храниться до шести символов. Целое число n

указывает ширину поля, отводимого для вывода содержимого одной ячейки. Если $n > 6$, то выводимый набор символов располагается в правых позициях поля, а избыточные левые позиции заполняются пробелами. Если $n \leq 6$, то выводятся только первые (левые) n символов из хранящихся в ячейке.

Пример. Участок программы с операторами вывода по спецификации A. (Цифра 5 перед спецификацией указывает число ее повторений см. 18.10).

```
DIMENSION TEXT (5)
DATA TEXT/'THERE IS NO MORE INFORMATION'/
PRINT 16, TEXT
PRINT 17, TEXT
PRINT 18, TEXT
16 FORMAT (5 A 8)
17 FORMAT (5 A 6)
18 FORMAT (5 A 4)
```

Результат печати:

```
—THERE— —IS—NO — —MORE—I—NFORMA—TION
THERE _IS_ NO_ MORE _INFORMATION
THERIS _NMORENFORTION
```

Чтобы понять результат печати, следует учесть, что в ячейке TEXT (1) хранятся символы THERE, в ячейке TEXT (2) символы IS_NO_, в ячейке TEXT (3) символы MORE_I, в ячейке TEXT (4) символы NFORMA и, наконец, в ячейке TEXT (5) хранятся символы TION_—. При выводе по спецификации A8 содержимое ячейки печатается в правой части поля из восьми позиций, т. е. дополняется слева двумя пробелами, что и видно на первой строке выдачи. По спецификации A6 содержимое ячейки совпадает с содержимым поля вывода. При выводе по спецификации A4 из ячейки берутся только первые четыре символа, т. е. из ячейки TEXT (1) выводятся символы THER, из ячейки TEXT (2) символы IS_N, из ячейки TEXT (3) символы MORE, из ячейки (4) символы NFOR и, наконец, из ячейки TEXT (5) выводятся символы TION, что и видно из третьей строки печати.

При вводе по спецификации A n поле ввода содержат n позиций. Символы, находящиеся в поле ввода, подвергаются ВКД-кодировке (см. приложение 1) и засылаются в соответствующие ячейки памяти машины согласно списку оператора ввода. Если n превосходит 6, то вводятся только 6 последних (правых) символов поля ввода. Если $n \leq 6$, то введенный набор символов помещается на место, указанное списком оператора ввода, и заполняет слева

направо соответствующую ячейку памяти. Избыточное место в ячейке заполняется пробелами.

I AM THE CAT WHO WALKES BY HIMSELF																																									
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

Рис. 28. Перфокарта ввода по спецификации А.

Пример. Будем вводить различными способами одну и ту же перфокарту, представленную на рис. 28. Пусть участок программы выглядит следующим образом:

```

DIMENSION CAT (6)
19 FORMAT (6A8)
20 FORMAT (6A6)
21 FORMAT (6A4)
22 FORMAT (6A6)
  READ 19, CAT
  PRINT 22, CAT
  READ 20, CAT
  PRINT 22, CAT
  READ 21, CAT
  PRINT 22, CAT

```

Результат печати:

```

AM — THEAT — WHOALKES — — HIMSE
I — AM — THE — CAT — WHO — WALKES — BY — HIMSELF
I — AM — — — THE — — — CAT — — — WHO — — — WAL — — KES

```

В этом примере печать ведется по спецификации A6, т. е. содержимое поля вывода совпадает с содержимым ячейки.

При вводе по спецификации A8 поле ввода состоит из восьми позиций, а вводятся только шесть последних символов. Так в первом поле ввода пробиты символы I _ AM _ THE, а содержимое ячейки CAT (1) есть AM _ THE. Во втором поле ввода стоят символы _ CAT _ WHO, а в ячейке CAT (2) содержится AT _ WHO. В третьем поле ввода стоит _ WALKES _, а CAT (3) содержит ALKES _. В четвертом поле ввода пробито BY _ HIMSE, а содержимое CAT (4) есть _ HIMSE. Наконец, пятое поле ввода содержит LF _ _ _ _ _ , а в ячейке CAT (5), так же как и в CAT (6), стоят одни пробелы.

При вводе по спецификации A6 содержимое ячейки совпадает с содержимым поля ввода.

При вводе по спецификации A4 поле ввода содержит четыре позиции. Символы, пробитые в этих позициях, помещаются в левую часть ячейки, а оставшееся в ней место заполняется двумя пробелами. В данном примере мы получаем:

Поле ввода	Содержимое	Ячейка	Содержимое
первое	I _ AM	CAT (1)	I—AM _ _
второе	_ THE	CAT (2)	_THE _ _
третье	_ CAT	CAT (3)	_CAT _ _
четвертое	_ WHO	CAT (4)	_WHO _ _
пятое	_ WAL	CAT (5)	_WAL _ _
шестое	KES _	CAT (6)	KES _ _

18.8. Масштабный коэффициент. При выводе чисел по спецификациям E и D перед десятичной точкой стоит одна цифра. Например, число —250,0067 при выводе по спецификации E15.6 печатается в виде

—2.500067+02

Иногда бывает более удобным, если на выдаче перед десятичной точкой стоит несколько цифр, например, если предыдущее число печатается в виде

—250.0067+00

Такая возможность имеется в языке фортран, и для ее реализации служит масштабный коэффициент sP.

Масштабный коэффициент sP применяется для изменения положения десятичной точки в формате вывода вещественных величин. Он может употребляться со спецификой F*n.m* при вводе и при выводе и со спецификациями E*n.m* и D*n.m* только при выводе. При вводе по спецификациям E и D масштабный коэффициент игнорируется. Спецификация с масштабным коэффициентом записывается следующим

образом:

$sPFn.m$

$sPEn.m$

$sPDn.m$

где P — символ, s — целая константа со знаком (знак $+$ можно опускать).

При вводе по спецификации F с масштабным коэффициентом sP вводимое число делится на 10^s и результат засылается в память машины. Например, если число 12566,4 вводится по спецификации $3PF7.0$, то после ввода в памяти машины хранится число

$$12566,4 \cdot 10^{-3} = 12,5664$$

При выводе по спецификации F с масштабным коэффициентом sP машинное число умножается на 10^s и результат выдается согласно спецификации F . Например, хранящееся в машине число 3,1415926536 можно выводить следующим образом:

Спецификация Результат печати

$F13.6$	3.141593
$1PF13.6$	31.415927
$3PF13.6$	3141.592654
$-1PF13.6$.314159

При выводе по спецификациям E и D с масштабным коэффициентом sP происходит сдвиг десятичной точки в формате вывода числа и соответственное изменение порядка числа. Например, то же число 3,1415926536 в зависимости от масштабного множителя имеет представления на выдаче:

Спецификация Результат печати

$E20.2$	3.14+00
$1PE20.2$	31.42-01
$2PE20.2$	314.16-02
$3PE20.2$	3141.59-03
$4PE20.2$	31415.93-04
$-1PE20.2$.31+01

Если масштабный множитель не указан, он предполагается равным нулю, однако, будучи указан один раз, он сохраняет силу для всех следующих за ним спецификаций F, E, и D в данном операторе FORMAT. Чтобы ликвидировать его влияние на последующие спецификации, перед одной из них следует поставить масштабный множитель OP. Например, рассмотрим последовательность операторов:

```
DIMENSION DATA (3)
DATA DATA/19.31, 0.014, 226.3/
PRINT 23, DATA
23 FORMAT (1PE15.2, E15.3, F10.2)
```

Выдача имеет вид

```
19.31 + 00      14.000 — 03      2262.00
      6 про-      3 про-
      белов      белая
```

Обратите внимание на изменение значения третьего элемента.

18.9. Управляющие и редакционные спецификации.

Управляющие спецификации помогают программисту удобно расположить информацию в поле ввода/вывода и сделать поясняющие надписи.

Спецификация nX используется при выводе для включения в единицу записи *n* пробелов, а при вводе для пропуска *n* символов.

Примеры. Пусть в ячейке KIND хранится число 6, в ячейке GAMEL хранится число 36,288, в ячейке HUMР хранится число 0,504. В результате работы операторов

```
WRITE (52, 547) KIND, CAMEL, HUMР
547 FORMAT (8X, I2, 1X, E9.3, 5X, F5.2)
```

мы получим следующую выдачу:

```
      63.629 + 01      0.50
      9 про-      6 про-
      белов      белов
```

Пусть значения переменных BAN, DAR и LOG набиты на перфокарте, как показано на рис. 29. Величины могут быть введены в память машины операторами:

```
READ 84, BAN, DAR, LOG
84 FORMAT (5X, F9.0, 5X, F10.0, 5X, 15)
```

Список спецификаций оператора FORMAT указывает, что значение переменной BAN пробито в позициях с 6-й по 14-ю, значение переменной DAR в позициях с 20-й по 29-ю, а значение переменной LOG пробито в позициях с 35-й по 39-ю. Позиции 1 — 5, 15 — 19 и 30 — 34 не включаются в поля ввода.

[illegible]

Рис. 29. Перфокарта ввода с использованием спецификации X.

Спецификация nH служит для ввода/вывода буквенно-цифровой информации и используется обычно для получения на выдаче всевозможных надписей. Здесь H — символ, признак типа спецификации, определяющий начало буквенно-цифрового поля; *n* — целая константа без знака, указывающая длину этого поля.

При выводе по спецификации nH происходит вывод всех символов, указанных в H -поле соответствующего оператора.

Примеры. В результате работы операторов

PRINT 5

5 FORMAT (10X, 16ННАЧАЛЬНЫЕ ДАННЫЕ)

печатается строка:

НАЧАЛЬНЫЕ ДАННЫЕ

11-я позиция

В результате работы операторов:

DATA X, F/8.5, —18 28/

PRINT 6, X, F

6 FORMAT (5X, 2HX = ,F5.2, 10X, 5HF(X) = , F7.3)

печатается строка:

X = 8.50

↑
6-я позиция

F(X) = — 18.280

↑
22-я позиция

При вводе по спецификации nH вводимая символьная информация заносится в H-поле (т. е. в следующие за символом H n позиций).

Пример. Рассмотрим работу операторов:

READ 7
7 FORMAT (20H DATA)
20 позиций

Вводимая перфокарта приведена на рис. 30.

Рис. 30. Перфокарта ввода по спецификации H.

После выполнения этих операторов оператор FORMAT с меткой 7 будет содержать последовательность символов, считанную

с двадцати первых позиций вводимой перфокарты. Ранее хранящийся текст стирается.

Если далее в программе появится оператор

PRINT 7

то будет отпечатана строка

13 DECEMBER 1972

Спецификация '...' является видоизменением спецификации *nH* и также служит для ввода/вывода символьной информации. В отличие от спецификации *nH*, в которой длина поля символов равна значению целой константы *n*, в спецификации '...' поле символов ограничивается апострофами. Например, запись

5 FORMAT (10X, 16ННАЧАЛЬНЫЕ ДАННЫЕ)

эквивалентна записи

5 FORMAT (10X, 'НАЧАЛЬНЫЕ ДАННЫЕ')

Символ / (слэш) в операторе FORMAT указывает на окончание физической единицы записи, т.е. осуществляет переход к новой строке или новой перфокарте. Например, оператор

PRINT 114

114 FORMAT (10X, 'INITIAL CONDITIONS', //, 1CX,
1HX, 16X, 'Y', /)

печатает следующие строки:

11-я позиция

INITIAL CONDITIONS 1 строка

2 строка

X

Y 3 строка

Пример.

READ 15, A, B, I, D
15 FORMAT (E10.3, F5.2/I8, E15.6)

Данный оператор FORMAT указывает на то, что значения переменных A, B, I, D набиты на двух перфокартах. Значение переменной A пробито на первой перфокарте в позициях с 1-й по 10-ю; значение переменной B пробито на первой перфокарте в позициях с 11-й по 15-ю; значение переменной I пробито на второй перфокарте в позициях с 1-й по 8-ю; значение переменной D пробито на второй перфокарте в позициях с 9-й по 23-ю.

18.10. Повторяемые спецификации. Любую спецификацию оператора **FORMAT** можно повторить, поставив перед спецификацией целую константу без знака, указывающую число повторений. Например, запись

FORMAT (I2, I2)

эквивалентна записи

FORMAT (2I2)

Если перед спецификацией имеется масштабный множитель **SP**, коэффициент повторения ставится между масштабным множителем и спецификацией. Например, запись

FORMAT (2PE12.3, E12.3)

эквивалентна записи

FORMAT (2P2E12.3)

Для повторения группы спецификаций эту группу следует заключить в скобки, а число повторений группы указать перед открывающей скобкой. Например, запись

FORMAT (E15.3, F6.1, I4, I4, E15.3, F6.1, I4, I4)

эквивалентна записи

FORMAT (2(E15.3, F6.1, 2I4))

Группа спецификаций оператора **FORMAT**, заключенных в скобки, называется *повторяемой группой*. Повторяемые группы могут образовывать гнезда глубиной до двух уровней. Например, рассмотрим оператор

FORMAT (F5.0/(2(F6.1, F7.2)/(F8.3)))

0 1 2 2 2 210

Скобки, помеченные нулем, соответствуют нулевому уровню; скобки, помеченные единицей, — первому уровню; скобки, помеченные двойкой, — второму уровню.

18.11. Оператор FORMAT. Ввод/вывод символьной информации, в отличие от ввода/вывода двоичных кодов, производится с помощью оператора

FORMAT, имеющего следующий вид:

FORMAT(<список спецификаций>)*

Здесь <список спецификаций> есть последовательность спецификаций преобразования и редакционных спецификаций, детально описывающих процесс обмена информацией между оперативной памятью машины и внешними устройствами.

Оператор FORMAT обязательно должен иметь метку. Оператор FORMAT может стоять в любом месте программы. На один и тот же оператор FORMAT могут ссылаться несколько операторов ввода/вывода.

Управление процессом ввода/вывода осуществляется списком спецификаций оператора FORMAT. Если не говорить о спецификациях с числом повторений, то список спецификаций читается слева направо. Каждой спецификации типа I, F, E, D, O, L, A ставится в соответствие один элемент в списке оператора ввода/вывода. Исключение представляет комплексный элемент, которому отвечают две спецификации преобразования типа F или E. Спецификациям типа X, H и '...' не ставится в соответствие элемент списка оператора ввода/вывода, а управление обменом передает соответствующую информацию непосредственно единице записи. Если в списке спецификаций встречается символ /, являющийся признаком начала новой единицы записи, то управление обменом заканчивает обработку текущей единицы записи и переходит к следующей. Например, при чтении с перфокарт символ / указывает на то, что дальнейшую информацию следует считывать со следующей перфокарты. При выдаче на печать символ / указывает на переход к новой строке.

Когда при чтении списка спецификаций оператора FORMAT управление обменом встречает спецификацию типа I, F, E, D, O, L или A, то проверяется, имеется ли элемент в списке операторе ввода/вывода. Если такой элемент имеется, то производится обмен информацией между оперативной памятью машины и внешним устройством, сопровождающийся требуемой спецификацией преобразованием этой информации. Если соответствующего элемента в списке опе-

ратора ввода/вывода нет, т.е. если этот список уже исчерпан, то обмен информацией заканчивается. Например, разберем последовательно работу оператора

PRINT 19, X

19 FORMAT (10X, 'X=', F7.2, 10X, 'Y=', F10.3)

1) Обрабатывается управляющая спецификация 10X и десять левых позиций строки выдачи заполняются пробелами;

2) обрабатывается спецификация '...': в одиннадцатой позиции ставится символ X, в двенадцатой позиции символ =;

3) управление попадает на спецификацию F. Происходит обращение к списку оператора PRINT, в котором находится элемент X. По идентификатору элемента из соответствующей ячейки памяти машины выбирается информация, преобразуется к символьному виду в согласии со спецификацией F и получившимися символами заполняются позиции с 13-й по 19-ю;

4) обрабатывается спецификация 10X. Позиции с 20-й по 29-ю заполняются пробелами;

5) обрабатывается спецификация '...'. В позициях 30-й и 31-й помещаются символы Y=;

6) обрабатывается спецификация E. Происходит обращение к списку оператора PRINT. Поскольку список исчерпан, оставшаяся часть строки заполняется пробелами и работа оператора PRINT заканчивается.

Если значение переменной X равнялось 11,326, то выдача имеет вид

$$\underbrace{\hspace{1.5cm}}_{10 \text{ пробелов}} X = 11.33 \underbrace{\hspace{1.5cm}}_{10 \text{ пробелов}} Y =$$

Если значение переменной X равнялось 11,326, то спецификаций доходит до последней закрывающей скобки оператора FORMAT, то обработка текущей единицы записи заканчивается и производится проверка, имеются ли еще необработанные элементы в списке оператора ввода/вывода. Если таких элементов нет, работа оператора заканчивается. Если список оператора ввода/вывода не исчерпан, то управление обменом начинается новую единицу записи и продолжает чтение

спецификаций, начиная с открывающей скобки первого уровня или, при отсутствии таковой с открывающей скобки нулевого уровня.

Примеры. Пусть одномерный массив А имеет 20 ячеек:
`DIMENSION A (20)`

и k -й элемент массива имеет значение K :

`DO 1 K = 1, 20`

`1 A(K) = K`

Будем печатать этот массив с помощью различных операторов `FORMAT`.

Оператор

`FORMAT (F5.0, 2(F6.1, F7.2), F8.3)`

эквивалентен оператору

`FORMAT (F5.0, (F6.1, F7.2, F6.1, F7.2), F8.3)`

$\begin{array}{ccccccc} & \uparrow & & \uparrow & & \uparrow & \uparrow \\ \text{нулевой} & & \text{первый} & & & & \text{нулевой} \\ \text{уровень} & & \text{уровень} & & & & \text{уровень} \end{array}$

и содержит скобки нулевого и первого уровня. Массив печатается в виде

1	2.0	3.00	4.0	5.00	6.000
7.0	8.00	9.0	10.00	11.000	
12.0	13.00	14.0	15.00	16.000	
17.0	18.00	19.0	20.00		

Обратите внимание на то, что чтение спецификаций для второй и последующих строк начинается со спецификации `F6.1`.

— Если скобки первого уровня убрать

`FORMAT (F5.0, F6.1, F7.2, F6.1, F7.2, F8.3)`

или ввести скобки второго уровня

`FORMAT ((F5.0, 2(F6.1, F7.2), F8.3))`

то выдача будет иметь вид

1	2.0	3.00	4.0	5.55	6.000
7	8.0	9.00	10.0	11.00	12.000
13	14.0	15.00	16.0	17.00	18.000
19	20.0				

— повторное чтение спецификаций ведется с начала списка.

Наличие в списке символа / приводит к следующим изменениям:

Оператор

`FORMAT (F5.0/2(F6.1, F7.2), F8.3)`

Выдача

1	2.0	3.00	4.0	5.00	6.000
7.0	8.00	9.0	10.00	11.000	
12.0	13.00	14.0	15.00	16.000	
17.0	18.00	19.0	20.00		

Оператор

FORMAT (F5.0/2(F6.1, F7.2)/F8.3)

Выдача

1			
2.0	3.00	4.0	5.00
6.000			
7.0	8.00	9.0	10.00
11.000			
12.0	13.00	14.0	15.00
16.000			
17.0	18.00	19.0	20.00

Наконец, если мы заключим последнюю спецификацию в скобки первого уровня

FORMAT (F5.0/2(F6.1, F7.2)/(F8.3)).

то получим в результате

1			
2.0	3.00	4.0	5.00
6.000			
7.000			
8.000			
9.000			
10.000			
11.000			
12.000			
13.000			
14.000			
15.000			
16.000			
17.000			
18.000			
19.000			
20.000			

18.12. Переменный список спецификаций оператора FORMAT. Выше мы рассматривали работу операторов ввода/вывода в случае фиксированного и не изменяющегося в процессе решения задачи списка спецификаций оператора FORMAT. При этом операторы ввода/вывода имели вид:

READ <i>n</i> ,	⟨список⟩
PRINT <i>n</i> ,	⟨список⟩
PUNCH <i>n</i> ,	⟨список⟩
READ (<i>i</i> , <i>n</i>)	⟨список⟩
WRITE (<i>i</i> , <i>n</i>)	⟨список⟩

где n — целая константа, метка оператора FORMAT, в соответствии со списком спецификаций которого ведется обмен информацией.

В фортране имеется и другой способ задания списка спецификаций преобразования. В операторе ввода/вывода вместо метки оператора FORMAT можно указать идентификатор переменной или массива переменных, в котором в виде текстовой константы хранится список спецификаций (вместе с открывающей и закрывающей список скобками). Например, оператор

READ 97, A

указывает на то, что чтение значения переменной (или массива переменных) A производится в соответствии со списком спецификаций оператора FORMAT, имеющего метку 97:

97 FORMAT (F5.0/2(F6.1, F7.2), F8.3)

Оператор

READ B, A

указывает на то, что чтение значения переменной (или массива переменных) A производится в соответствии со списком спецификаций, который в виде символов составляет значение переменной B.

Список спецификаций (вместе с левой и правой скобками) может быть присвоен переменной либо с помощью оператора DATA, либо вводом по спецификации A.

Например, пусть в программе имеются следующие операторы:

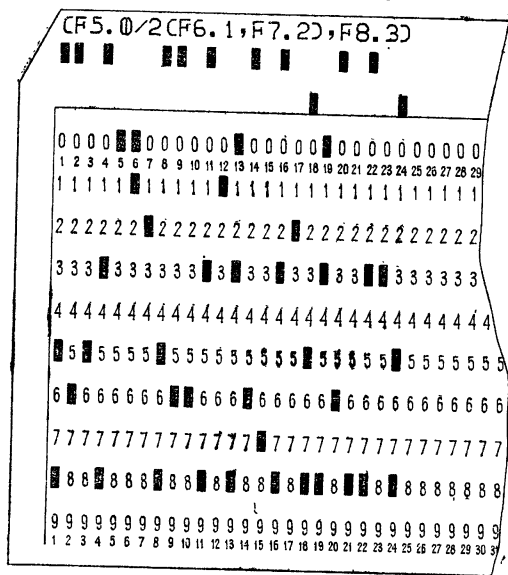
```
DIMENSION A(10), B(10)
DATA(B=24H(F5.0/2(F6.1, F7.2), F8.3))
. . . . .
PRINT B, A
```

Печать значений массива A ведется в соответствии со списком спецификаций

(F5.0/2(F6.1, F7.2), F8.3)

PRINT B, A

Приведем еще один пример использования переменного списка спецификаций. Пусть содержимое



```
1 FORMAT (E12.2, F8.2, 2I7)
```


а операторы

K=4

PRINT L(K), A, B, I, J

эквивалентны операторам

PRINT 2, A, B, I, J

2 FORMAT (F8.2, E12.2, 2I7)

§ 19. Функции и подпрограммы

19.1. Основная программа и подпрограммы.

В практике программирования часто встречается ситуация, когда некоторый блок операторов повторяется в программе несколько раз. Чтобы не переписывать такой блок многократно, его можно поместить где-нибудь в конце программы и при необходимости обращаться к нему с помощью операторов передачи управления. Для упрощения подобного рода переходов в фортране предусмотрены функции-подпрограммы и подпрограммы. Функции-подпрограммы и подпрограммы представляют собой блоки операторов, транслируемые независимо.

Та программа, с выполнения которой начинается решение задачи, называется *основной программой*. Основная программа может обращаться к подпрограммам. Основная программа, так же как и подпрограммы, имеет правила оформления. Первым оператором основной программы должен быть оператор

PROGRAM <идентификатор>

где <идентификатор> является последовательностью от одного до шести буквенно-цифровых символов, первый из которых — буква.

Последним из операторов, относящихся к данной программе, должен быть оператор END. Оператор END служит концом последовательности операторов, принадлежащих программе, так что в основную программу входят все операторы, заключенные между операторами PROGRAM и END.

Пример оформления программы. Программа вводит массив чисел a_1, a_2, \dots, a_n , вычисляет их среднее $b = \sum_{i=1}^n a_i/n$ и печатает это значение.

```

PROGRAM MEAN
DIMENSION A (100)
READ 1, N
1 FORMAT (I5)
READ 2, (A(I), I = 1, N)
2 FORMAT (8 F10.0)
B = 0.0
DO 3 I = 1, N
3 B = B + A(I)
B = B/N
PRINT 4, B
4 FORMAT (10X, 'AVERAGE =' E 10.3)
END

```

Основную программу, функцию-подпрограмму или подпрограмму называют *программной единицей*. Все программные единицы транслируются независимо друг от друга, а связь между ними устанавливается только на время решения задачи с помощью специальной программы-загрузчика. Последовательность операторов, относящихся к программной единице, называют *телом* программной единицы. Соответственно говорят о теле программы или подпрограммы. Первым оператором программной единицы должен стоять оператор, указывающий ее вид и имя. Например, оператор

PROGRAM VERA

указывает, что программная единица является основной программой и имеет имя VERA.

Последним оператором программной единицы должен быть оператор END.

19.2. Библиотечные функции. Некоторые простые функции (типа синуса, экспоненты) хранятся в машине в библиотеке стандартных программ. Каждая из этих функций имеет свое имя — идентификатор (см. приложение 2). Для вычисления такой функции достаточно, чтобы в выражении появилось ее имя с указанием в скобках значения аргумента. Например, для вычисления комплексного числа $z = r(\cos \varphi + i \sin \varphi)$ при значении модуля $r = 2$ и угла $\varphi = \pi/4$ в программе достаточно написать

COMPLEX Z

$Z = 2. * (1., 0) * \text{COS}(3.14/4.) + (0., 1.) * \text{SIN}(3.14/4.)$

Встретив такой оператор, машина разделит 3,14 на 4,

обратится с полученным результатом к функциям вычисления синуса и косинуса, вернется в основную программу, произведет арифметические действия и присвоит получившееся значение выражения переменной Z.

При обращении к функции ее аргументом может быть константа, переменная, переменная с индексом или любое другое выражение. В частности, это выражение может само содержать обращение к функциям.

Примеры обращения к библиотечным функциям:

TN = COS (N * ATAN(SQRT(1. - X * X)/X))

DLOG = ALOG(ABS(1. - DZETA))/DZETA

XMAX = AMAX 1(X(1), X(2), X(3))

X1 = (-B + SQRT (B ** 2 - 4. * A * C))/(2. * A)

19.3. Функция-оператор. Функция-оператор позволяет организовать удобное обращение к выражению, зависящему от параметров.

Например, в программе несколько раз приходится вычислять комплексную экспоненту $z = e^{x+iy}$. Мы можем определить функцию Z(X, Y):

COMPLEX. Z

Z(X, Y) = EXP(X) * (SOC(Y) + (0., 1) * SIN(Y))

и далее в программе обращаться к этой функции просто посредством указания ее идентификатора и параметров. Так, для вычисления $v = 1/2e^{1+i}$ достаточно написать

$$V = Z(1., 1.)/2.$$

В общем случае *функция-оператор* имеет вид

$$F(P_1, P_2, \dots, P_k) = E$$

Здесь F — идентификатор, название функции-оператора; P_1, P_2, \dots, P_k — идентификаторы аргументов функции, $1 \leq k \leq 63$; E — выражение.

На идентификатор функции-оператора накладываются обычные условия. Он должен состоять не более чем из шести символов, каждый из которых является буквой или цифрой, причем первый символ обязательно должен быть буквой.

Тип функции-оператора может быть указан в операторе описания типа (см. предыдущий пример). Если

тип функции-оператора не описан, то он определяется ее идентификатором: идентификатор, начинающийся с букв I, J, K, L, M, N, указывает на функцию целого типа; идентификаторы, начинающиеся с других букв алфавита, — на функцию типа вещественная.

Наименование функции-оператора не должно появляться в операторах DIMENSION, EQUIVALENCE, COMMON или EXTERNAL данной программной единицы.

Если функция-оператор имеет одинаковое имя с какой-либо библиотечной функцией или функцией-подпрограммой, то в процессе счета обращение будет происходить к функции-оператору.

Например,

$$\text{COS}(X) = 1. - X**2/2.$$

.

$$Y = X * \text{COS}(0.5 * X)$$

При вычислении последнего выражения программа обратится не к библиотечной функции COS, а к определенной ранее функции-оператору.

Аргументы P_1, P_2, \dots, P_n , перечисленные в скобках при определении функции-оператора, называются *формальными параметрами*. Формальные параметры должны быть простыми переменными. Влияние идентификаторов формальных параметров распространяется лишь на выражение функции-оператора. Так, в предыдущем примере формальный параметр X, указанный при определении функции-оператора COS, не имеет никакого отношения к переменной X, встречающейся в программе.

Аргументы, указываемые при обращении к функции-оператору, называются *фактическими параметрами*. Для вычисления значения функции-оператора в ее выражение вместо формальных параметров представляются значения фактических параметров и выполняются операции, указанные этим выражением. Между формальными и фактическими параметрами должно быть соответствие в количестве, типах и порядке их следования. Каждый фактический параметр может быть выражением, т. е. константой, переменной (простой или с индексом), функцией или комбинацией этих величин, разделенных знаком операций и скобками.

Выражение E, стоящее в определении функции-оператора, может быть арифметическим или логическим выражением. Оно не должно содержать переменных с индексами. Выражение E может обращаться к библиотечным функциям, к ранее определенным функциям-операторам и функциям-подпрограммам. В выражении могут встречаться переменные, не являющиеся формальными параметрами. Значения таких переменных должны быть определены в программе до обращения к функции-оператору.

Функции-операторы располагаются в программе перед первым исполняемым оператором, но после всех неисполняемых операторов типа DIMENSION и т. д. (см. п. 16.1).

Примеры записи функций-операторов

$$\text{SUM}(W, X) = W * \text{EXTERN}(C1 + C2 * X)$$
$$F(X) = X ** 2 / (\text{EXP}(X) - 1.)$$
$$\text{ONE}(A, B) = (A . \text{OR} . B) . \text{AND} . \text{NOT} . B . \text{OR} . A . \text{AND} . B$$

Прокомментируем первую запись. Функция-оператор SUM имеет два аргумента. Формальные параметры обозначены через W и X. В выражение функции-оператора входят также две переменные C1 и C2, значения которых будут определены в программе до обращения к функции. Функция-оператор SUM сама обращается к функции EXTERN, имеющей один аргумент. Для функции EXTERN выражение $C1 + C2 * X$ служит фактическим параметром.

19.4. Функция-подпрограмма. Функция-подпрограмма позволяет удобным образом оформить блок операторов, целью работы которого является вычисление значения одной переменной. В отличие от функции-оператора, где значение переменной получается в результате вычисления одного выражения, функция-подпрограмма может содержать любое число операторов.

Обращение к функции-подпрограмме аналогично обращению к библиотечной функции или к функции-оператору: в выражении указывается идентификатор функции и вслед за идентификатором в скобках через запятую перечисляются значения аргументов функции — фактические параметры.

Функция-подпрограмма представляет собой блок операторов, оформленный специальным образом и транслируемый независимо от основной программы и других подпрограмм, т. е. от других программных еди-

ниц. Связь между программными единицами устанавливается только на время решения задачи.

Первым оператором функции-подпрограммы должен быть один из следующих операторов:

FUNCTION F(P₁, P₂, ..., P_k)
REAL FUNCTION F(P₁, P₂, ..., P_k)
INTEGER FUNCTION F(P₁, P₂, ..., P_k)
DOUBLE PRECISION FUNCTION F(P₁, P₂, ..., P_k)
COMPLEX FUNCTION F(P₁, P₂, ..., P_k)
LOGICAL FUNCTION F(P₁, P₂, ..., P_k)

Здесь F — идентификатор, наименование функции; P₁, P₂, ..., P_k — идентификаторы, формальные параметры, $1 \leq k \leq 63$.

Перечисленные операторы указывают:

- 1) начало блока операторов, относящихся к функции-подпрограмме;
- 2) идентификатор — имя функции;
- 3) тип функции, т. е. тип ее результата.
- 4) параметры — аргументы функции.

Идентификатор функции может содержать от одного до шести символов, каждый из которых должен быть буквой или цифрой, причем первым символом должна быть буква.

Если началом функции-подпрограммы служит оператор

FUNCTION F(P₁, P₂, ..., P_k)

то тип функции определяется ее идентификатором: идентификаторы, начинающиеся с букв I, J, K, L, M, N, указывают на функцию целого типа. В противном случае тип функции считается вещественным. Если тип функции определен операторами REAL FUNCTION, INTEGER FUNCTION и т. д., то в каждой вызывающей программной единице тип функции должен быть указан в операторе описания типа. Например, если тело функции-подпрограммы имеет вид

COMPLEX FUNCTION Z(R, PHI)

Z=R*((1., 0.)*COS(PHI)+(0., 1.)*SIN(PHI))

RETURN

END

то использование этой функции в другой программной единице может выглядеть следующим образом:

COMPLEX Z, SUMM

SUMM=Z(SQRT(2.), .7854)+(1., 0.)

Идентификатор функции не должен упоминаться в неисполняемых операторах, относящихся к функции-подпрограмме, исключая сам оператор FUNCTION. В результате работы функции-подпрограммы переменной, имеющей ее название, должно быть присвоено некоторое значение. Это можно сделать либо оператором присваивания, либо оператором ввода, либо в результате обращения к другой подпрограмме.

Аргументы функции-подпрограммы, указываемые в операторе FUNCTION, называются *формальными параметрами*. Число формальных параметров не должно превосходить 63, но присутствие хотя бы одного параметра обязательно. Все формальные параметры должны быть идентификаторами. Это могут быть идентификаторы простых переменных, идентификаторы массивов переменных или идентификаторы других функций-подпрограмм и подпрограмм. Формальные параметры служат для указания типа, количества и порядка записи фактических параметров, которые ставятся им в соответствие при каждом обращении к функции-подпрограмме. Идентификаторы формальных параметров имеют силу лишь для операторов, принадлежащих данной функции-подпрограмме, а вне этой функции те же идентификаторы могут иметь иной смысл.

Ни один из формальных параметров не должен упоминаться в операторах EQUIVALENCE, DATA, EXTERNAL или COMMON в функции-подпрограмме.

Формальные параметры, являющиеся идентификаторами массивов, должны быть описаны в операторе DIMENSION или ему эквивалентном внутри функции-подпрограммы. При этом, если размеры массива также являются формальными параметрами, допускаются массивы переменной длины. Например, функция-подпрограмма, вычисляющая сумму диагональ-

ных элементов матрицы, может иметь следующий вид:

```
FUNCTION SPUR (A, N)
  DIMENSION A (N, N)
  SUMM = 0.0
  DO 1 K = 1, N
1 SUMM = SUMM + A(K, K)
  SPUR = SUMM
  RETURN
END
```

Если тип формального параметра не определяется идентификатором, его тип должен быть описан внутри функции-подпрограммы.

Тело функции-подпрограммы составляют все операторы, заключенные между операторами FUNCTION и END. Таким образом, END указывает на окончание последовательности операторов, относящихся к данной подпрограмме. То место в функции-подпрограмме, где при ее выполнении следует прекратить вычисления и вернуться в вызывающую программу, указывается оператором RETURN. Одна подпрограмма может содержать несколько операторов RETURN, но присутствие хотя бы одного из них обязательно.

Например, функция

$$\text{sign } x = \begin{cases} 1 & \text{при } x > 0, \\ 0 & \text{при } x = 0, \\ -1 & \text{при } x < 0 \end{cases}$$

может быть оформлена следующим образом:

```
FUNCTION SIGN (X)
  IF (X) 1, 2, 3
1 SIGN = -1.0
  RETURN
2 SIGN = 0.0
  RETURN
3 SIGN = 1.0
  RETURN
END
```

Функция-подпрограмма может иметь несколько входов (см. п. 19.7. Оператор ENTRY).

Все идентификаторы, появляющиеся в функции-подпрограмме, за исключением названия функции и

идентификаторов, указанных в операторе COMMON (см. п. 19.8), имеют локальный характер и могут независимо использоваться в других программных единицах.

Функция-подпрограмма не должна содержать операторов, которые прямо или косвенно обращаются к ней самой.

Аргументы функции-подпрограммы, указываемые при обращении к ней, называются *фактическими параметрами*. Между формальными и фактическими параметрами должно быть соответствие в количестве, типах и порядке их следования. При обращении к функции-подпрограмме в качестве фактических параметров допускаются:

1) выражение, т. е. константа, переменная (простая или с индексом), функция или комбинация этих величин, разделенных знаками операций и скобками;

2) идентификатор массива переменных;

3) идентификатор другой функции-подпрограммы или подпрограммы.

Если формальный параметр функции-подпрограммы определяется или переопределяется в ней, то соответствующий ему фактический параметр должен быть простой переменной, переменной с индексом или идентификатором массива переменных.

Если фактический параметр является идентификатором функции-подпрограммы или подпрограммы, этот идентификатор должен быть описан в операторе EXTERNAL в вызывающей программе (см. п. 19.6. Оператор EXTERNAL).

19.5. Подпрограмма. Подпрограмма является блоком операторов, оформленным специальным образом, составляющим программную единицу и транслируемым независимо. В отличие от функции-подпрограммы, цель работы которой состоит в вычислении значения одной переменной — самой функции, результатом работы подпрограммы может быть вычисление любого числа переменных, в частности ни одной.

Первым оператором подпрограммы должен быть один из следующих операторов:

SUBROUTINE S

SUBROUTINE S(P₁, P₂, ..., P_k)

Здесь S — идентификатор, наименование подпрограммы; P_1, P_2, \dots, P_k — идентификаторы, формальные параметры.

Оператор SUBROUTINE определяет:

1) начало блока операторов, относящихся к подпрограмме;

2) название подпрограммы;

3) параметры подпрограммы.

Идентификатор S подпрограммы, как и любой другой идентификатор, может содержать от одного до шести буквенно-цифровых символов, первым из которых является буква. Для подпрограмм не определяется понятие типа, т. е. не имеет смысла говорить о целой подпрограмме, логической подпрограмме и т. д. В результате работы подпрограммы ее идентификатору не присваивается никакого значения.

Идентификатор подпрограммы не должен упоминаться в неисполняемых операторах подпрограммы, за исключением самого оператора SUBROUTINE. Он не должен также упоминаться внутри подпрограммы в операторах присваивания, в операторах ввода/вывода и при обращении к другим программным единицам.

Аргументы подпрограммы, указываемые в операторе SUBROUTINE, называются *формальными параметрами*. Их число не должно превосходить 63. Все формальные параметры должны быть идентификаторами. Это могут быть идентификаторы простых переменных, идентификаторы массивов переменных или идентификаторы других подпрограмм и функций подпрограмм.

Формальные параметры служат для указания типа, количества и порядка записи фактических параметров, которые ставятся им в соответствие при каждом вызове подпрограммы. Идентификаторы формальных параметров имеют силу лишь для операторов, принадлежащих данной подпрограмме, и вне этой подпрограммы те же идентификаторы могут иметь иной смысл.

Ни один из формальных параметров не должен упоминаться в операторах EQUIVALENCE, DATA, EXTERNAL или COMMON внутри подпрограммы.

Формальные параметры, являющиеся идентификаторами массивов, должны быть описаны в операторе

DIMENSION или эквивалентном ему внутри подпрограммы. При этом, если размеры массива также являются формальными параметрами, допускаются массивы переменной длины.

Пример оформления программы перемножения двух прямоугольных матриц $C = A \cdot B$. Матрица A имеет M строк и N столбцов, матрица B имеет N строк и M столбцов, матрица C — квадратная размера $M \times M$. Элементы матрицы C вычисляются

$$\text{по формуле } c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}.$$

```
SUBROUTINE MPROD (A, B, C, M, N)
  DIMENSION A(M, N), B(N, M), C(M, M)
  DO 1 I = 1, M
    DO 1 J = 1, M
      C(I, J) = 0.0
      DO 1 K = 1, N
        1 C(I, J) = C(I, J) + A(I, K) * B(K, J)
      RETURN
    END
```

Если тип формального параметра не определяется идентификатором, его тип должен быть описан внутри подпрограммы.

Тело подпрограммы составляют все операторы, заключенные между операторами **SUBROUTINE** и **END**. Таким образом, **END** указывает на окончание последовательности операторов, относящихся к данной подпрограмме. То место в подпрограмме, где при ее выполнении следует прекратить вычисления и вернуться в вызывающую программу, указывается оператором **RETURN**. Одна подпрограмма может содержать несколько операторов **RETURN**, но присутствие хотя бы одного из них обязательно.

Подпрограмма может иметь несколько входов (см. п. 19.7. Оператор **ENTRY**).

Все идентификаторы, появляющиеся в подпрограмме, за исключением названия подпрограммы и идентификаторов, перечисленных в операторе **COMMON** (см. п. 19.8), имеют локальный характер и могут использоваться независимо в других программных единицах.

Подпрограмма не должна прямо или косвенно обращаться к самой себе.

Обращение к подпрограмме осуществляется оператором CALL, имеющим вид

CALL S

либо

CALL S(Q₁, Q₂, ..., Q_k)

Здесь S — идентификатор, имя вызываемой подпрограммы; Q₁, Q₂, ..., Q_k — аргументы подпрограммы, фактические параметры.

Оператор CALL передает управление подпрограмме, и она выполняется до тех пор, пока в ней не встретится оператор RETURN, который осуществляет обратную передачу управления в вызывающую программу на оператор, следующий за оператором CALL.

Между формальными и фактическими параметрами должно быть соответствие в количестве, типах и порядке их следования. При обращении к подпрограмме в качестве фактических параметров допускаются:

- 1) выражение, т. е. константа, переменная (простая или с индексом), функция или комбинация этих величин, разделенных знаками операций и скобками;
- 2) идентификатор массива переменных;
- 3) идентификатор другой подпрограммы или функции-подпрограммы.

Если формальный параметр подпрограммы определяется или переопределяется в ней, то соответствующий ему фактический параметр должен быть простой переменной, переменной с индексом или идентификатором массива переменных.

Если фактический параметр является идентификатором функции-подпрограммы или подпрограммы, то этот идентификатор должен быть описан в операторе EXTERNAL в вызывающей программе (см. п. 19.6. Оператор EXTERNAL).

Пример обращения к описанной выше подпрограмме произведения матриц:

```
PROGRAM MATRIX
DIMENSION U(5, 4), V(4, 5), W(5, 5)
. . .
CALL MPROD (U, V, W, 5, 4)
```

После выполнения оператора CALL в массиве W содержится матричное произведение U на V.

19.6. Оператор EXTERNAL*). Если при обращении к некоторой функции-подпрограмме или к подпрограмме фактический параметр является идентификатором другой функции-подпрограммы или подпрограммы, то этот идентификатор должен быть описан в операторе EXTERNAL. В противном случае он будет рассматриваться как идентификатор переменной.

Пример. Функция-подпрограмма SIMPS вычисляет значение интеграла $\int_a^b f(x) dx$ методом Симпсона по $2n$ точкам:

$$\int_a^b f(x) dx \approx$$

$$\approx \frac{h}{3} (f_0 + 4f_1 + 2f_2 + 4f_3 + \dots + 2f_{2n-2} + 4f_{2n-1} + f_{2n}),$$

где

$$h = \frac{b-a}{2n}; \quad f_i = f(x_i); \quad x_i = a + ih.$$

Функция-подпрограмма SIMPS должна обращаться к функции-подпрограмме F, вычисляющей значения подынтегральной функции. Поскольку эта функция в различных задачах может иметь разные идентификаторы, удобно оформить подпрограмму таким образом, чтобы идентификатор F был формальным параметром.

```
FUNCTION SIMPS (A, B, N, F)
  H = 0.5 * (B - A) / N
  ODD = 0.0 3EVEN = 0.0
  DO 10 K = 1, N
    X = A + (2 * K - 1) * H
    ODD = ODD + F(X)
  10 EVEN = EVEN + F(X + H)
  SIMPS = H/3. * (F(A) + 4. * ODD + 2. * EVEN - F(B))
  RETURN
END
```

Если функция подпрограммы SIMPS оформлена таким образом, то для вычисления $\int_0^\pi \sin x dx$ по 10 точкам достаточно написать следующие операторы:

EXTERNAL SIN

Y = SIMPS(0., 3.14159, 5, SIN)

*) От английского слова external — внешний.

Значение переменной Y после выполнения этих операторов будет равно приближенному значению интеграла.

Если подынтегральная функция не является библиотечной, то вычисление ее должно быть оформлено как функция-подпрограмма. Например, для вычисления $\int_0^1 \varphi(x) dx$, где $\varphi(x) = e^{-x^2}$, следует оформить функцию

```
FUNCTION PHI(X)
PHI=EXP(-X**2)
RETURN
END
```

после чего вычисление интеграла по $2k$ точкам осуществляется операторами

```
EXTERNAL PHI
```

```
      . . . . .
A=SIMPS(0., 1., K, PHI)
```

В общем случае оператор EXTERNAL имеет вид

```
EXTERNAL F1, F2, ..., Fk
```

где F_i — идентификаторы функций-подпрограмм или подпрограмм. Подчеркнем еще раз, что оператор EXTERNAL должен находиться в вызывающей программе. Оператор EXTERNAL является неисполняемым оператором и должен предшествовать первому исполняемому оператору программной единицы.

19.7. Оператор ENTRY*). Этот оператор обеспечивает дополнительные входы в программную единицу. Он имеет вид

```
ENTRY Q
```

где Q — идентификатор данного входа. Идентификатор оператора ENTRY по своим функциям полностью эквивалентен идентификатору, указанному в операторах SUBROUTINE или FUNCTION. В случае функции-подпрограммы идентификаторы дополнительных

*) От английского слова entry — вход.

входов должны иметь тот же тип, что и идентификатор основного входа.

Пример. Значения функций Бесселя $J_0(x)$ и $J_1(x)$ с точностью до двух-трех знаков можно вычислять по формулам

$$J_0(x) = \frac{1}{3} \sum_{j=1}^3 \cos(x \cdot z_j),$$

$$J_1(x) = \frac{1}{3} \sum_{j=1}^3 z_j \sin(x \cdot z_j),$$

где

$$z_1 = 0.965963, \quad z_2 = 0.707107, \quad z_3 = 0.258819.$$

Оформим функцию-подпрограмму:

```

FUNCTION BESJ(X)
  DIMENSION Z(3), CS(3)
  DATA Z/.965963, .707107, .258819/
  ENTRY BESJ0
  DO 10 J = 1, 3
10  CS(J) = COS(X * Z(J))
  GO TO 30
  ENTRY BESJ1
  DO 20 J = 1, 3
20  CS(J) = Z(J) * SIN(X * Z(J))
30  W = 0.0
  DO 40 J = 1, 3
40  W = W + CS(J)
  BESJ = W/3.
  RETURN
END

```

В этой подпрограмме есть два дополнительных входа BESJ0 и BESJ1. Обращение к подпрограмме посредством выражения BESJ0(X) приводит к вычислению $J_0(x)$. Выражение BESJ1(X) вычисляет $J_1(x)$. Например, для вычисления $J_2(x) = \frac{2}{x} J_1(x) - J_0(x)$ в программе можно записать

$$\text{BESJ2} = 2. * \text{BESJ1}(X)/X - \text{BESJ0}(X)$$

Оператор ENTRY может располагаться в любом месте подпрограммы, исключая область действия оператора DO. Оператор ENTRY не должен иметь метки. Число входов в одну программную единицу не должно превосходить 20. При оформлении функции-подпрограммы результат вычисления должен быть при-

своей переменной, имя которой указано в операторе FUNCTION. Для входа в подпрограмму в месте, указанном оператором ENTRY, в вызывающей программе нужно использовать идентификатор данного входа, после которого указать все фактические параметры подпрограммы в том порядке, как это определено операторами SUBROUTINE или FUNCTION.

19.8. Оператор COMMON. Мы уже неоднократно подчеркивали, что программные единицы (основная программа, функции-подпрограммы и подпрограммы) транслируются независимо друг от друга и вследствие этого идентификаторы и метки, встречающиеся в них, являются локальными параметрами, имеющими смысл лишь для операторов данной программной единицы. В другой программе те же метки и идентификаторы могут использоваться независимо. Исключение составляют только идентификаторы самих программных единиц, имеющие глобальное значение. С одной стороны, такая ситуация удобна, поскольку не надо заботиться о том, чтобы метки и идентификаторы в различных программных единицах были различными. Но, с другой стороны, это затрудняет работу с переменными, общими для нескольких программных единиц.

В языке фортран допускаются блоки глобальных переменных, доступные для нескольких программных единиц. Все глобальные (общие) переменные должны быть описаны в операторе COMMON, имеющем вид

COMMON <список>/Q₁/<список>/Q₂/<список> ...

Здесь Q_i — идентификаторы, наименования блоков общих переменных, <список> есть последовательность идентификаторов переменных или массивов, перечисленных через запятую. Эти величины составляют содержание общего блока, наименование которого указано в предшествующих списке слэшах /.../.

Наименование блока может отсутствовать, как у первого списка в приведенном выше операторе. Блок COMMON, имеющий наименование, называется *помеченным блоком*. При отсутствии наименования блок называется *непомеченным*. Непомеченный общий блок характеризуется либо отсутствием идентификатора

блока и связанных с ним слэшей (если он стоит в начале оператора COMMON), либо наличием двух последовательных слэшей // перед списком блока. Например,

COMMON A, B, C

— один непомеченный блок, состоящий из трех элементов.

COMMON//A, B, C

— оператор эквивалентен предыдущему.

COMMON/BLOCK1/A, B/CDE/C(10), D(10, 10),
E(10, 10)

— два помеченных блока. Блок BLOCK1 состоит из двух переменных. Блок CDE состоит из 210 переменных.

В списках блоков могут указываться идентификаторы простых переменных и массивов (с индексами или без индексов). Если идентификатор массива переменных в списке блока не имеет индексов, определяющих его размер, то размер массива должен быть определен оператором DIMENSION либо оператором описания типа в той же программной единице.

Например, последовательность операторов

COMMON/ONE/P, RHO, T
COMPLEX T(5)
DIMENSION P(3, 4, 2)

эквивалентна последовательности

COMMON/ONE/P(3, 4, 2), RHO, T(5)
COMPLEX T

Оператор COMMON позволяет программисту резервировать участки машинной памяти, которыми могут пользоваться несколько программных единиц. Каждому такому участку присваивается название — идентификатор блока. Идентификаторы различных общих блоков не должны совпадать. Непомеченный блок тоже должен быть только один. Соответствие между общими переменными в различных программных единицах устанавливается по идентификаторам блоков, а не по идентификаторам переменных, указанных в списке блока.

Например, начало программы имеет вид

```
PROGRAM BETA  
COMMON/TWO/BRA, KET//DELTA  
DIMENSION KET(2, 3), DELTA(8)  
DOUBLE PRECISION BRA(3)
```

Описание указывает, что в программе, имеется два блока COMMON: помеченный блок TWO и непомеченный блок. Блок TWO имеет длину 12 ячеек. Непомеченный блок имеет длину 8 ячеек. Оператор COMMON устанавливает, что DELTA(1) есть первая ячейка непомеченного блока, DELTA(2) — вторая ячейка и т. д. Поэтому, если далее в функции-подпрограмме будет указано

```
FUNCTION LION (V, W)  
COMMON FIX(5), BRA(3)
```

то это означает, что в данной программной единице первой ячейке непомеченного блока COMMON присвоено название FIX(1), второй FIX(2) и т. д. Отсюда вытекает следующее соответствие между элементами массива DELTA основной программы и элементами массивов FIX и BRA функции-подпрограммы LION:

DELTA(1)	FIX(1)
DELTA(2)	FIX(2)
DELTA(3)	FIX(3)
DELTA(4)	FIX(4)
DELTA(5)	FIX(5)
DELTA(6)	BRA(1)
DELTA(7)	BRA(2)
DELTA(8)	BRA(3)

Из схемы видно, что значение переменной DELTA(6) в основной программе рассматривается как значение переменной BRA(1) в функции LION и оба эти наименования относятся к одной и той же ячейке памяти машины. Отметим, что идентификатор BRA используется в качестве имени различных переменных в основной программе и в подпрограмме.

О соответствии элементов общих блоков следует помнить во избежание ошибок.

Идентификаторы общих блоков могут совпадать с идентификаторами переменных и массивов (но не с идентификаторами программных единиц). Например, допускается запись

COMMON/A/A(10)

где идентификатор А использован и как наименование общего блока и как наименование массива переменных.

Две переменные одного блока COMMON или переменные различных блоков нельзя описывать как эквивалентные (см. п. 14.5. Оператор EQUIVALENCE). Однако допускается установление эквивалентности элементов общих блоков с локальными переменными программных единиц. В этом случае, если переменная, которой приписывается эквивалентность элементу блока COMMON, является элементом массива переменных, то автоматически возникает соответствие между остальными элементами этого массива и элементами блока COMMON. Такое соответствие может изменить длину общего блока, но оно не должно изменять его начало. Например,

COMMON A, B, C
DIMENSION D(3)
EQUIVALENCE (B, D(1))

— допустимая последовательность операторов. Элемент D(1) эквивалентен элементу В общего блока. Элемент D(2) эквивалентен элементу С. Элемент D(3) увеличивает длину общего блока. Последовательность операторов

COMMON A, B, C
DIMENSION D(3)
EQUIVALENCE (B, D(3))

недопустима, поскольку она меняет начало общего блока.

19.9. BLOCK DATA. Мы уже говорили в п. 14,6, что в операторе DATA программной единицы нельзя присваивать значения переменным, входящим в COMMON блоки. По правилам фортрана засылка

данных оператором DATA в непомеченный общий блок вообще не допускается. Для засылки данных в переменные из помеченного блока COMMON оформляется специальная программная единица BLOCK DATA. Эта программная единица начинается оператором BLOCK DATA и заканчивается, как обычно, оператором END. В ней не должно быть исполняемых операторов, т. е. она должна содержать только операторы COMMON, DATA, DIMENSION и операторы описания типа переменных, входящих в общие блоки.

19.10. Соответствие массивов программных единиц. В заключение параграфа остановимся на вопросе о соответствии элементов массивов в программных единицах.

Если формальный параметр подпрограммы является идентификатором массива, то размер этого массива должен быть описан внутри подпрограммы. Размер массива — фактического параметра описывается в другой программной единице, в вызывающей программе, и вследствие этого размеры массивов формального и фактического параметров могут не совпадать. На самом деле массивов, определенных формальными параметрами, в памяти машины не существует. Подпрограмма производит действие над массивами фактических параметров. Но положение определенного элемента массива относительно начала вычисляется в подпрограмме по описанию массива — формального параметра.

Например, пусть в вызывающей программе и подпрограмме массивы описаны следующим образом:

```
PROGRAM BIBL
  REAL MULT (3, 2)
  . . . . .
  CALL SUB180 (MULT)
  . . . . .
  END
  SUBROUTINE SUB180 (PROD)
  DIMENSION PROD (2, 3)
  . . . . .
  RETURN
  END
```

При работе подпрограммы SUB180 преобразования, производящиеся над элементами матрицы PROD, будут фактически производиться над элементами матрицы MULT. Однако из-за различных описаний этих массивов элемент PROD(I, J) не обязательно соответствует элементу MULT(I, J). Соответствие матриц PROD и MULT устанавливается только по их первому элементу, а соответствие остальных элементов возникает автоматически с учетом способа хранения массивов в памяти машины (см. п. 14.3). Поскольку матрицы хранятся по столбцам, получаем следующее соответствие:

MULT (1,1)	PROD (1, 1)
MULT (2, 1)	PROD (2, 1)
MULT (3, 1)	PROD(1, 2)
MULT (1, 2)	PROD (2, 2)
MULT (2, 2)	PROD (1, 3)
MULT (3, 2)	PROD (2, 3)

У одномерных массивов всегда получается правильное соответствие элементов. Например, если фактический параметр описан как

DIMENSION VECTOR(20)

то при любом из описаний формального параметра SCALAR

DIMENSION SCALAR(20)
 DIMENSION SCALAR(1)
 DIMENSION SCALAR(100)
 DIMENSION SCALAR(N)

элемент VECTOR(I) соответствует элементу SCALAR(I).

У двумерных массивов для правильного соответствия элементов достаточно совпадения длин столбцов, т. е. максимальных значений первого индекса. Для соответствия элементов трехмерных массивов в описаниях фактического и формального параметров должны совпадать максимальные значения первого и второго индексов.

Отметим, что у формального и фактического параметров может быть нарушено соответствие не только элементов массивов, но и их размерностей. Например, если упомянутую выше подпрограмму SUB180 оформить как

```
SUBROUTINE SUB 180 (PROD)
  DIMENSION PROD (1)
```

```
  . . . . .
```

```
  RETURN
```

```
  END
```

то при обращении к подпрограмме

```
CALL SUB180 (MULT)
```

между элементами фактического параметра матрицы MULT размера 3×2 и элементами вектора PROD установится следующее соответствие:

MULT (1, 1)	PROD (1)
MULT (2, 1)	PROD (2)
MULT (3, 1)	PROD (3)
MULT (1, 2)	PROD (4)
MULT (2, 2)	PROD (5)
MULT (3, 2)	PROD (6)

Все сказанное выше о соответствии между элементами фактических и формальных параметров относится также и к соответствию между элементами блока COMMON в различных программных единицах.

1. КОДИРОВКА СИМВОЛОВ

Символ	Внутренний Код Данных (ВКД)	Код УПП	Код CDC	Код IBM (ЕС ЭВМ)
A	101	040	12,1	12,1
B	102	042	12,2	12,2
C	103	061	12,3	12,3
D	104	077	12,4	12,4
E	105	045	12,5	12,5
F	106	100	12,6	12,6
G	107	101	12,7	12,7
H	110	055	12,8	12,8
I	111	102	12,9	12,9
J	112	103	11,1	11,1
K	113	052	11,2	11,2
L	114	104	11,3	11,3
M	115	054	11,4	11,4
N	116	105	11,5	11,5
O	117	056	11,6	11,6
P	120	060	11,7	11,7
Q	121	106	11,8	11,8
R	122	107	11,9	11,9
S	123	110	0,2	0,2
T	124	062	0,3	0,3
U	125	111	0,4	0,4
V	126	112	0,5	0,5
W	127	113	0,6	0,6
X	130	065	0,7	0,7
Y	131	063	0,8	0,8
Z	132	114	0,9	0,9
0	060	000	0	0
1	061	001	1	1
2	062	002	2	2
3	063	003	3	3
4	064	004	4	4
5	065	005	5	5
6	066	006	6	6
7	067	007	7	7
8	070	010	8	8
9	071	011	9	9

Символ	Внутренний Код Данных (ВКД)	Код УПП	Код CDC	Код IBM (ЕС ЭВМ)
/	057	014	0, 1	0, 1
+	053	012	12	12, 6
—	055	013	11	11
пробел	040	017	пробел или	пробел или
.	056	016	0, 2, 8	0, 2, 8
)	051	023	12, 3, 8	12, 3, 8
	044	127 *)	12, 4, 8	11, 5, 8
\$	052	031	11, 3, 8	11, 3, 8
,	054	015	11, 4, 8	11, 4, 8
(050	022	0, 3, 8	0, 3, 8
=	075	025	0, 48	12, 5, 8
:	047	031 (032)	3, 8	6, 8
:	072	037	4, 8	5, 8
Б	142	041	12, 5, 8	2, 8
Г	147	043	11, 0	12, 11, 0, 2, 8
Д	144	044	11, 2, 8	12, 11, 0, 7, 8
Ж	166	046	12, 6, 8	12, 11, 0, 4, 8
З	172	047	11, 8, 9	11, 0, 4, 8, 9
			0, 8, 9	2, 11, 0, 2, 8, 9
И	151	050	0, 6, 8	12, 0, 3, 8, 9
И	152	051	2, 8	12, 0, 4, 8, 9
Л	154	053	11, 8, 6	12, 0, 6, 8, 9
П	160	057	0, 8, 7	12, 11, 4, 8, 9
Ф	146	064	12, 0, 8	12, 11, 0, 6, 8
Ц	143	066	11, 0, 8	12, 11, 0, 3, 8
Ч	176	067	0, 1, 8	12, 11, 0, 6, 8,
Ш	173	070	12, 8, 2	9
Щ	175	071	9, 8	12, 11, 0, 3, 8
Ы	171	072	11, 1, 8	9
Ь	170	073	12, 8, 9	12, 11, 0, 5, 8,
Э	174	074	12, 1, 8	9
Ю	140	075	1, 8	11, 0, 7, 8, 9
Я	161	076	5, 8	11, 0, 6, 8, 9
				12, 11, 0, 4, 8,
				9
				12, 11, 0, 8
				12, 11, 5, 8, 9

*) В коде УПП в качестве заменителя символа \$ выбран символ .

Пояснения.

1. Внутренний код данных восьмиразрядный. Первая цифра кодов соответствует 8- и 7-му разрядам, вторая цифра 6—4-му разрядам, третья цифра 3—1-му разрядам.

2. Код УПП тоже восьмиразрядный. Первая цифра приведенных в таблице кодов соответствует 7-му разряду, вторая цифра 6—4-му разрядам, третья цифра 3—1-му разрядам. Восьмой разряд добавляет число единиц в коде до нечетного значения.

3. В обозначениях кодов для поколонной набивки на перфокартах (кодах CDC и IBM) цифры обозначают номера строк. Строки располагаются сверху вниз: 12, 11, 0, 1, 2, ..., 9.

2. ОТЛИЧИЯ ФОРТРАНА-ДУБНА ОТ ФОРТРАНА-IV

1. **Константы.** Минимальная единица информации, которую обрабатывают машины серии IBM/360 и ЕС ЭВМ, называется **байтом**. Байт состоит из восьми разрядов или битов. Для хранения значений констант и переменных в памяти машины отводятся группы байтов (поля) различной длины:

слово — это группа из четырех последовательных байтов (32 разряда или бита);

полуслово — это группа из двух байтов (16 разрядов);

двойное слово — группа из восьми байтов (64 разряда).

1.1 *Целая константа* в фортране-IV занимает одно слово и поэтому по абсолютной величине не должна превосходить 2147483647 ($2^{31} - 1$).

1.2. *Вещественная константа может* занимать одно слово или двойное слово. Во втором случае она аналогична константе с двойной точностью и при записи, так же как и константа с двойной точностью, характеризуется символом D, стоящим перед порядком. Абсолютная величина вещественной константы может быть нулем или лежать в пределах от 16^{-65} (приблизительно 10^{-78}) до 16^{63} (приблизительно 10^{75}). Константа, занимающая одно слово, имеет мантиссу в 6 шестнадцатиричных цифр (приблизительно 7 десятичных цифр), а константа, занимающая двойное слово, имеет мантиссу в 14 шестнадцатиричных цифр (приблизительно 17 десятичных цифр).

1.3. *Комплексные константы* должны состоять из вещественных констант одного типа, т. е. все вещественные константы должны занимать либо по четыре байта, либо по восемь байтов.

1.4. *Текстовые константы.* Число символов в текстовой константе не должно превосходить 255 (в отличие от 120 в фортране-Дубна).

1.5. *Шестнадцатиричные константы.* Вместо восьмеричных констант, используемых в фортране-Дубна, в фортране-IV имеются шестнадцатиричные константы. Шестнадцатиричная константа представляет собой символ Z, за которым следует шестнадцатиричное число, формируемое из цифр от 0 до 9 и из букв от A до F. Поскольку байт содержит две шестнадцатиричные цифры, константа, имеющая нечетное число цифр, дополняется спереди нулем.

Приводим внутренний код шестнадцатиричных цифр:

0 — 0000	8 — 1000
1 — 0001	9 — 1001
2 — 0010	A — 1010
3 — 0011	B — 1011
4 — 0100	C — 1100
5 — 0101	D — 1101
6 — 0110	E — 1110
7 — 0111	F — 1111

Примеры. Константа Z1C49A2F1 имеет двоичное представление

00011100010010011010001011110001

Константа ZBADFADE имеет двоичное представление

0000101110101101111101011011110

Здесь первые четыре нуля вставлены, поскольку в константе нечетное число цифр.

Максимальное количество цифр, допускаемых в шестнадцатиричной константе, зависит от количества байтов, отводимых для ее хранения.

Длина поля
(байты)

Максимальное число
шестнадцатиричных цифр

16	32
8	16
4	8
2	4
1	2

Если число цифр в константе превосходит максимально допустимое, то лишние левые шестнадцатиричные цифры отбрасываются. Если число цифр меньше максимального, слева к константе добавляются нули.

2. Переменные. В фортране-IV символ \$ считается буквой и допускается наравне с другими буквами в идентификаторах переменных. Например, допускается идентификатор \$VAR.

2.1. Типы переменных. В фортране-IV различают четыре типа переменных: *целые, вещественные, комплексные и логические*. Для переменной каждого типа имеются стандартная и допустимая длины, определяющие количество байтов машинной памяти, отводимое для хранения значения переменной.

Тип переменной	Стандартная длина (в байтах)	Допустимая длина (в байтах)
INTEGER	4	2
REAL	4	8
COMPLEX	8	16
LOGICAL	4	1

Для переменных, тип которых не описан в программе, действует стандартное соглашение:

если идентификатор переменной начинается с букв I, J, K, L, M, N, то она считается целой и имеет длину 4 байта;

если первая буква идентификатора отлична от указанных, переменная считается вещественной длины 4 байта.

Тип переменной может быть описан либо в операторе *неявного* (implicit) описания, либо в операторе *явного* (explicit) описания типа.

Оператор IMPLICIT неявного описания типа изменяет стандартное соглашение о типе переменной и позволяет приписать тип и длину группе переменных по начальной букве их идентификаторов.

Пример 1.

IMPLICIT REAL(A — H, O — \$), INTEGER(I — N)

Все переменные, идентификаторы которых начинаются с букв от I до N, объявляются целыми стандартной длины 4 байта. Все переменные, идентификаторы которых начинаются с букв от A до H, от O до Z и с \$, считаются вещественными переменными стандартной длины 4 байта. Приведенный оператор IMPLICIT соответствует стандартному соглашению о типах переменных.

Пример 2.

IMPLICIT INTEGER * 2(A — H), REAL * 8(I — K),
LOGICAL (L, M, N)

Все переменные, идентификаторы которых начинаются с букв от A до H, объявляются целыми переменными длины 2 байта. Переменные с идентификаторами, начинающимися с букв от I до K, объявляются вещественными переменными длины 8 байтов. Все идентификаторы, начинающиеся с букв L, M, N, считаются идентификаторами логических переменных стандартной длины 4 байта. Для идентификаторов, начальные буквы которых не указаны в операторе IMPLICIT, т. е. идентификаторов, начинающихся с букв от O до Z и с \$, действует стандартное соглашение. Все они принадлежат вещественным переменным длины 4 байта.

Пример 3.

IMPLICIT COMPLEX * 16(C — F)

Идентификаторы, начинающиеся с букв от C до F, объявляются идентификаторами комплексных переменных, у которых под действительную и мнимую части отводится по 8 байтов. Для идентификаторов, начинающихся с букв A, B, от G до Z, и с \$, действуют стандартные соглашения.

Оператор IMPLICIT должен быть первым оператором в основной программе (в фортране-IV нет оператора PROGRAM) и вторым оператором в функции-подпрограмме и подпрограмме.

Операторы явного описания типа устанавливают тип переменной или массива переменных по идентификатору этой переменной, а не по начальной букве идентификатора. Кроме того, в оператор описания типа можно указать длину переменной, размерность массива и задать начальные данные. Таким образом,

в фортране-IV операторы описания типа отчасти берут на себя не только функции оператора DIMENSION, но и оператора DATA. Имеется четыре оператора описания типа: INTEGER, REAL, COMPLEX, LOGICAL.

Пример 1.

INTEGER * 2 ITEM/76/, VALUE

Оператор указывает, что переменные ITEM и VALUE являются целыми и под каждую из них отводится память в два байта. Кроме того, переменной ITEM присваивается начальное значение 76.

Пример 2.

COMPLEX C, D/(2.1, 4.7)/, E * 16

Оператор указывает, что переменные C, D и E являются комплексными. Переменные C и D имеют стандартную длину 8 байтов (4 байта под действительную часть и 4 байта под мнимую часть), и переменной D присваивается начальное значение (2.1, 4.7). Переменная E имеет длину 16 байтов.

Пример 3.

REAL * 8 BAKER, HOLD, VALUE * 4, ITEM(5,5)

Оператор указывает, что переменные BAKER, HOLD, VALUE и массив ITEM имеют вещественный тип. ITEM является идентификатором матрицы размера 5×5 . Переменные BAKER и HOLD занимают по 8 байтов каждая. Переменная VALUE имеет длину 4 байта. Содержимое массива ITEM хранится в 200 байтах (8 байтов на каждый элемент массива).

Пример 4.

REAL A(5,5)/20 * 6.9E2,5 * 1.0/, B(100)/100 *

TEST * 8(5)/5 * 0.0D0/

Оператор указывает, что A, B, TEST — массивы вещественных переменных. Массив A занимает 100 байтов (по 4 байта на каждый элемент массива). Первым двадцати элементам массива A присваивается начальное значение 6.9E2, а последним пяти элементам — начальное значение 1.0. Массив B занимает 400 байтов (по 4 байта на каждый элемент массива). Всем элементам массива B присваивается начальное значение 0.0. Массив TEST занимает 40 байтов памяти машины (по 8 байтов на элемент). Каждой переменной массива TEST присваивается начальное значение 0.0D0.

В фортране-IV имеется также оператор описания типа DOUBLE PRECISION. Этот оператор эквивалентен подобному оператору в дубненской версии языка фортран. В фортране-IV он аналогичен оператору REAL * 8, но не позволяет присваивать переменным начальные значения.

2.2. *Массивы переменных.* В отличие от фортрана-дубна, где максимальная размерность массивов переменных равна трем, в фортране-IV допускаются массивы, имеющие размерность вплоть до семи.

В качестве индексов переменных массивов допускаются произвольные арифметические выражения (целого или вещественного

типа), которые могут содержать знаки арифметических операций $+$, $-$, $*$, $/$, $**$, скобки, обращения к функциям и элементы других массивов. Выражения смешанного типа вычисляются по обычным правилам и в случае вещественного результата он преобразуется в целый.

2.3 *Оператор DATA* имеет вторую из форм, описанных в 14.6.

3. **Выражения.** Класс допустимых арифметических выражений в фортране-IV шире, чем в фортране-Дубна. Знаки операций $+$, $-$, $*$, $/$ могут быть связаны переменные любых типов. Допускается выражение $A ** B ** C$, которое вычисляется как $(A ** B) ** C$. Показатель степени для вещественного или целого основания должен быть вещественным или целым. Показатель степени для комплексного основания должен быть целым.

4. **Операторы.** В фортране-IV не допускается запись нескольких операторов на одной строке через символ $\$$.

Арифметический оператор присваивания $A = E$ может связывать переменную A и выражение E любых типов.

5. **Операторы ввода и вывода информации.**

5.1. *Оператор READ.* Оператор ввода в фортране-IV в общем случае имеет следующий вид:

READ (*a*, *b*, ERR*c*, END*d*) <список>

где a — целая константа без знака, либо целая переменная длины 4, указывающая номер канала обмена информацией; b (может быть опущено) — либо метка оператора **FORMAT**, либо идентификатор переменной (массива), содержащей список спецификаций преобразования, либо идентификатор **NAMelist**; $ERR = c$ (может быть опущено) — c — метка оператора, на который надо передать управление в случае возникновения ошибки при работе оператора; $END = d$ (может быть опущено), d — метка оператора, на который нужно передать управление в случае, если массив данных ввода исчерпан.

Основными формами оператора ввода служат:

READ (*a*, *b*) <список>

ввод по каналу a с преобразованием информации в соответствии либо со списком спецификаций оператора **FORMAT** (если b метка), либо списком, содержащемся в переменной b (если b идентификатор);

READ (*a*) <список>

бесформатный ввод по каналу a ;

READ (*a*, *x*)

ввод с использованием **NAMelist**.

5.2. *Оператор WRITE.* Оператор вывода в фортране-IV в общем случае имеет вид

WRITE (*a*, *b*) <список>

где a — целая константа без знака либо целая переменная длины 4, указывающая номер канала обмена информацией; b (может быть опущено) — либо метка оператора **FORMAT**, либо идентификатор переменной (массива), содержащей список спецификаций преобразования, либо идентификатор **NAMelist**,

Подобно оператору READ оператор WRITE имеет три основные формы:

WRITE(*a*, *b*)<список> — форматный вывод;
WRITE(*a*)<список> — бесформатный вывод;
WRITE(*a*, *x*) — вывод с использованием NAMELIST.

Операторы

READ*n*, <список>
PRINT*n*, <список>
PUNCH*n*, <список>

в фортрэне-IV также допускаются.

5.3. Ввод и вывод с использованием NAMELIST. Оператор NAMELIST используется в связи с операторами ввода/вывода READ(*a*, *x*) и WRITE(*a*, *x*) и позволяет вводить и выводить информацию без выписывания списка. Оператор NAMELIST присваивает имя *x* списку идентификаторов переменных или массивов.

Оператор имеет вид

NAMELIST/*x*₁/*<список>*/*x*₂/*<список>*...

где *x*₁, *x*₂, ... — идентификаторы, <список> — последовательность идентификаторов переменных и массивов.

Для ввода с использованием NAMELIST данные должны иметь специальную форму записи. Первая позиция на каждой единице записи (например, перфокарте) должна быть пробелом. Во второй позиции на первой единице записи должен стоять символ &, за которым выписывается без пробелов идентификатор NAMELIST. За идентификатором должен быть хотя бы один пробел, а потом через запятую перечисляются элементы вводимой информации, конец которой указывается символами & END. Каждый элемент информации имеет вид

<имя> = <список констант>

где <имя> есть идентификатор переменной, переменная с индексом, или идентификатор массива переменных, а <список констант> есть константа либо последовательность констант, перечисленных через запятую. Идентификаторы элементов вводимой информации должны быть указаны в списке NAMELIST, причем порядок их несуществен.

Пример. Пусть *A* — одномерный массив из трех элементов, а *I* и *L* — матрицы 3×3 .

NAMELIST/NAM1/*A*, *B*, *I*, *J*, *L*/NAM2/*C*, *J*, *I*, *L*
READ (5, NAM1)
WRITE (6, NAM2)

Оператор NAMELIST определяет два списка: NAM1 и NAM2. Оператор READ вводит значения переменных списка NAM1 по пятому каналу. Предположим, что вводимые карты имеют следующий вид:

Вторая позиция
Первая карта &NAM1 *I* (2,3) = 5, *J* = 4, *B* = 3. 2
Последняя карта *A* (3) = 4.0, *L* = 2, 3, 7 * 4, & END

Читается первая карта. Проверяется совпадение идентификатора с идентификатором NAMELIST, указанным в операторе READ (если идентификаторы не совпадают, читается первая карта следующей группы NAMELIST). Целые константы 5 и 4 присваиваются переменным I(2,3) и J соответственно; действительные константы 3.2 и 4.0 присваиваются переменным B и A(3). Поскольку идентификатор массива L указан без индексов, массив заполняется константами последовательно. Целые константы 2 и 3 помещаются в L(1,1) и L(2,1), а целая константа 4 присваивается переменным L(3,1), L(1,2), ..., L(3,3).

Оператор WRITE выводит значения переменных, перечисленных в списке NAM2, по шестому каналу. Предположим, что значения J, L и I(2,3) не изменились, что C присвоено значение 428.0E+03, что I(1,3) равно 6, а все остальные элементы массива I имеют нулевое значение. Тогда, если происходит перфорация, получим

Вторая позиция

Первая карта	& NAM2
Вторая карта	C = 428000 00, J = 4, I = 0, 0, 0, 0, 0, 6.5
Третья карта	0, L = 2, 3, 4, 4, 4, 4, 4, 4
Четвертая карта	& END

5.4 *Некоторые дополнительные отличия.* В фортране-IV имеются операторы ввода/вывода прямого доступа. Мы не будем останавливаться на этих операторах, поскольку в фортране-Дубна нет их аналога.

В фортране-IV нет операторов ENCODE и DECODE.

6. **Спецификация формата.** В фортране-IV определены следующие спецификации преобразования: I, F, E, D, Z, G, L, A. Спецификации O в фортране-IV нет.

6.1. *Спецификация Z* используется для ввода/вывода шестнадцатеричных цифр. Аналогична спецификации O фортрана-Дубна.

6.2. *Спецификация G* является обобщенной спецификацией для преобразования целых, вещественных и логических величин и имеет черты спецификаций I, E, F, D или L в зависимости от типа переменной, участвующей в обмене информацией.

6.3. *Редакционная спецификация T* используется для задания позиции, в которой нужно расположить начало передаваемой информации.

7. **Функции и подпрограммы.**

7.1. *Оператора PROGPAM* в фортране-IV нет.

7.2. *Оператор RETURNi.* В фортране-IV можно возвращаться из подпрограммы не на оператор, непосредственно следующий за оператором CALL, а на любой оператор, имеющий метку. Для этого:

а) в качестве формального параметра допускается символ *;
б) соответствующий фактический параметр должен иметь вид &n, где n — метка оператора, на который происходит возврат из подпрограммы;

в) в качестве оператора возврата допускается оператор RETURNi, где i — целая константа или целая переменная длины 4, значение которой, скажем, k, указывает k-ю метку в списке параметров оператора SUBROUTINE.

Пример. Вызывающая программа

```
. . . . .  
10 CALL SUB (A B, C, & 30, & 40)  
20 Y = A + B  
  
. . . . .  
30 Y = A + C  
  
. . . . .  
40 Y = B + C  
  
. . . . .  
END
```

Подпрограмма

```
SUBROUTINE SUB (X, Y, Z, *, *)  
. . . . .  
100 IF (M) 200, 300, 400  
200 RETURN  
300 RETURN 1  
400 RETURN 2  
END
```

Оператор с меткой 10 в вызывающей программе передает управление подпрограмме SUB. После выполнения в ней оператора с меткой 100 возврат в основную программу произойдет на операторы с метками 20, 30, 40 в зависимости от того, отрицательно, равно нулю или положительно значение переменной M.

3. БИБЛИОТЕЧНЫЕ ФУНКЦИИ-ПОДПРОГРАММЫ

Название функции	Математическое обозначение	Обозначение в фортране	Число аргументов	Тип аргументов	Тип функции
Экспонента	e^x	EXP (X)	1	веществ.	веществ.
Натуральный логарифм	$\ln x$	DEXP (X)	1	двойн. точн. комплекс.	двойн. точн. комплекс.
		CEXP (X)	1	комплекс.	комплекс.
		ALOG (X)	1	веществ.	веществ.
		DLOG (X)	1	двойн. точн. комплекс.	двойн. точн. комплекс.
Десятичный логарифм	$\lg x$	CLOG (X)	1	веществ.	веществ.
		ALOG10(X)	1	веществ.	веществ.
Синус	$\sin x$	DLOG10 (X)	1	двойн. точн. веществ.	двойн. точн. веществ.
		SIN (X)	1	веществ.	веществ.
		DSIN (X)	1	двойн. точн. комплекс.	двойн. точн. комплекс.
Косинус	$\cos x$	CSIN (X)	1	комплекс.	комплекс.
		COS (X)	1	веществ.	веществ.
		DCOS (X)	1	двойн. точн. комплекс.	двойн. точн. комплекс.
Корень квадратный	\sqrt{x}	CCOS (X)	1	комплекс.	комплекс.
		SQRT (X)	1	веществ.	веществ.
		DSQRT (X)	1	двойн. точн. комплекс.	двойн. точн. комплекс.
		CSQRT (X)	1	комплекс.	комплекс.
Абсолютное значение (модуль)	$ x $	ABS (X)	1	веществ.	веществ.
		IABS (X)	1	целый	целый
		DABS (X)	1	двойн. точн. комплекс.	двойн. точн. комплекс.
		CABS (X)	1	комплекс.	комплекс.
Арктангенс *)	$\operatorname{arctg} x$	ATAN (X)	1	веществ.	веществ.
		DATAN (X)	1	двойн. точн. веществ.	двойн. точн. веществ.
		ATAN 2(X, Y)	2	веществ.	веществ.
	$\operatorname{arctg} \left(\frac{x}{y} \right)$	DATAN 2(X, Y)	2	двойн. точн.	двойн. точн.

Название функции	Математическое обозначение	Обозначение в фортране	Число аргументов	Тип аргументов	Тип функции
Тангенс гиперболический	$\text{th } x$	TANH (X)	1	веществ.	веществ.
Остаток **)	$x \pmod{y}$	AMOD (X, Y)	2	веществ.	веществ.
		MOD (X, Y)	2	целые	целые
		DMOD (X, Y)	2	двойн. точн.	двойн. точн.
Целая часть аргумента ***)	$[x]$	AINT (X)	1	веществ.	веществ.
		INT (X)	1	целая	целая
Выбор максимального значения	$\max(x_1, x_2, \dots, x_n)$	IDINT (X)	1	двойн. точн.	двойн. точн.
		AMAX 0	≥ 2	целые	веществ.
		AMAX 1	≥ 2	веществ.	веществ.
		MAX 0	≥ 2	целые	целая
		MAX 1	≥ 2	целые	целая
		DMAX 1	≥ 2	двойн. точн.	двойн. точн.
Выбор минимального значения	$\min(x_1, x_2, \dots, x_n)$	AMIN 0	≥ 2	целые	веществ.
		AMIN 1	≥ 2	веществ.	веществ.
		MIN 0	≥ 2	целые	веществ.
		MIN 1	≥ 2	целые	целая
		DMIN 1	≥ 2	двойн. точн.	двойн. точн.
Присвоение знака величине, знак второго аргумента присваивается абсолютной величине первого аргумента	$\text{sign } y \cdot x $	SIGN (X, Y) ISIGN (X, Y) DSIGN (X, Y)	2 2 2	веществ. целые двойн. точн.	веществ. целая двойн. точн.

Название функции	Математическое обозначение	Обозначение в формате	Число аргументов	Тип аргументов	Тип функции
Положительная разность	$x - \min(x, y)$	DIM (X, Y)	2	веществ.	веществ.
Плавающая запятая		IDIM (X, Y)	2	целые	целая
Преобразование целой величины в вещественную		FLOAT	1	целый	веществ.
Фиксированная запятая		IFIX	1	веществ.	целая
Преобразование вещественной величины в целую		SNGL	1	двойн. точн.	веществ.
Преобразование вещественной величины с двойной точностью в вещественную величину		DBLE	1	двойн. точн.	веществ.
Преобразование вещественной величины с двойной точностью в вещественную величину		REAL	1	комплекс.	веществ.

Название функции	Математическое обозначение	Обозначение в форме	Число аргументов	Тип аргументов	Тип функции
Выделение мнимой части у комплексной величины		AIMAG	1	комплекс.	веществ.
Объединение двух вещественных величин в комплексную величину		SMPLX (X, Y)	2	веществ.	комплекс.
Получение комплексно-сопряженной величины		CONJG (Z)	1	комплекс.	комплекс.

*) Функция ATAN принимает значения в интервале $(-\pi/2, +\pi/2)$, т. е. если $Y=ATAN(X)$, то $x=\lg y$, где $-\pi/2 < y < \pi/2$. Функция ATAN 2 принимает значения в интервале $(-\pi, \pi)$ и определена таким образом, что если $Z=ATAN 2(X, Y)$, то $\sin z = x / \sqrt{x^2 + y^2}$, $\cos z = y / \sqrt{x^2 + y^2}$, т. е. аргументы X и Y являются катетами треугольника, имеющего искомый угол. В частности, выражение ATAN 2 (0., 0.) приводит к сообщению об ошибке ERROR NUMBER 356 выражение ATAN 2 (0., Y) равно нулю при $Y > 0$ выражение ATAN 2 (± 0 , Y) равно $\pm \pi$ при $Y < 0$. Аналогично определяется функция DATAN 2, отличающаяся от функции ATAN 2 тем, что ее аргумент и значение являются величинами с двойной точностью.

**) Остаток $x \pmod{y}$ определяется как $x - [x/y] \cdot y$, где $[]$ обозначают целую часть числа. Например, $2 \pmod{3} = 2$, а $3 \pmod{2} = 1$.

***) Целая часть $[x]$ определяется как целое число, имеющее знак аргумента и не превосходящее по абсолютной величине абсолютной величины аргумента. Например, $[2.5] = 2$, $[-2.5] = -2$.

ОТВЕТЫ К УПРАЖНЕНИЯМ

§ 2.

1.

8.	—1.16E—11	347.4	—2.8E—5
552.6	1.E+5		3.14159

2. Вторая и шестая константы вещественные.

3. Вторая константа целая.

§ 3. Идентификаторы целых переменных:

IP2 IFIP MARIA KAPPA MADLEN

Идентификаторы вещественных переменных:

ARRAY BETA2 DRUM DELTA

AJAX HUME A ALGOL COBOL

§ 4.

1. $A + B * X + C * X ** 2$
2. $A + X * (B + C * X)$
3. $(A + B * X) ** 3$
4. $A + B * X / (C + D * X)$
5. $\text{SQRT}((1. - \cos(X))/2.)$
6. $\text{SQRT}(P * (P - A) * (P - B) * (P - C))$
7. $\text{ALOG}(\text{ABS}((1. - X)/(1. + X)))$

§ 5.

1. $P7 = (429. * X ** 7 - 693. * X ** 5 + 315. * X ** 3 - 35. * X) / 16.$
2. $Y = A + B/X + C/X ** 2$
3. $Y = (A * X ** 2 + B * X + C) / X ** 2$
4. $F = 1. / (A * X ** 2 + B)$
5. $X = \text{ALOG}(-A * B ** 2 / C / D ** 2) / (D - B)$
6. $P = A * \text{EXP}(-(B ** 2 + 2. * C) / (4. * C))$
7. $U = \text{EXP}(-A * X) * \sin(\text{OMEGA} * X + \text{PHI})$

§ 7.

1. READ 1, A, B, C
 ZSQ = SQRT (B**2 - 4. * A * C)
 X1 = (-B + ZSQ)/(2. * A)
 X2 = (-B - ZSQ)/(2. * A)
 PRINT 2, A, B, C, X1, X2
 1 FORMAT (E15.3)
 2 FORMAT (E20.3)
2. READ 1, R, H
 A = 2. * SQRT (2. * H * R - H ** 2)
 PRINT 2, R, H, A
 1 FORMAT (E15.3)
 2 FORMAT (E20.3)
3. READ 1, R, H1, H2
 PI = 3.14159
 Z = SQRT (R ** 2 + ((H2 - H1)/2.) ** 2)
 S = PI * R * (H1 + H2 + R + Z)
 V = PI * R ** 2 * (H1 + H2)/2.
 PRINT 2, R, H1, H2, S, V
 1 FORMAT (E15.3)
 2 FORMAT (E20.3)
4. READ 1, X
 Y = ATAN (SQRT (1. - X * X)/X)
 PRINT 2, X, Y
 1 FORMAT (E15.3)
 2 FORMAT (E20.3)
5. READ 1, X, A
 U = ALOG((X + A) ** 3/(X ** 2 - A * X + A ** 2))
 V = ATAN (X * SQRT (3.)/(2. * A - X))
 W = U/2. + SQRT (3.) * V
 PRINT 2, X, A, U, V, W
 1 FORMAT (E15.3)
 2 FORMAT (E20.3)

§ 9.

- 1a) XMAX = X1
 IF (XMAX - X2) 10, 20, 20
 10 XMAX = X2
 20 IF (XMAX - X3) 30, 40, 40
 30 XMAX = X3
 40 CONTINUE
- 6) XMIN = X1
 IF (XMIN - X2) 10, 10, 20
 20 XMIN = X2
 10 IF (XMIN - X3) 30, 30, 40
 40 XMIN = X3
 30 CONTINUE

```

b)      IF (X2 - X1) 10, 20, 20
        10 ZX = X1
           X1 = X2
           X2 = ZX
        20 IF (X3 - X1) 30, 40, 40
        30 ZX = X1
           X1 = X3
           X3 = ZX
        40 IF (X3 - X2) 50, 60, 60
        50 ZX = X2
           X2 = X3
           X3 = ZX
        60 CONTINUE

r)      INTX = X
        IF (X) 10, 20, 20
        10 INTX = INTX - 1
        20 CONTINUE

2.      PI2 = 2. * 3.14159
        10 IF (TETA) 20, 30, 30
        20 TETA = TETA + PI2
           GO TO 10
        30 IF (TETA - PI2) 40, 50, 50
        50 TETA = TETA - PI2
           GO TO 30
        40 CONTINUE

3.      HNM = 2. * X
        HN = 4. * X ** 2 - 2.
        DO 10 N = 3, 4
        HNP = 2. * X * HN - 2. * N * HNM
        HNM = HN
        10 HN = HNP

4.      XS = 0.0
        10 EXPXS = EXP (-XS)
           IF (ABS (XS - EXPXS) - 1.E - 4) 30, 20, 20
        20 XS = XS - (XS - EXPXS)/(1. + EXPXS)
           GO TO 10
        30 CONTINUE

```

§ 10.

```

1.      SCAL = 0.0
        DO 10 I = 1, N
        10 SCAL = SCAL + A (I) * B (I)

2.      SPUR = 0.0
        DO 20 K = 1, M
        20 SPUR = SPUR + ENERGY (K, K)

3.      DO 30 I = 1, L
        DO 30 J = 1, N
        W (I, J) = 0.0
        DO 30 K = 1, M
        30 W (I, J) = W (I, J) + U (I, K) * V (K, J)

```

```

4.    DISTAN = 0.0
      DO 40 I = 1, N
40    DISTAN = DISTAN + (X(I) - Y(I)) ** 2
      DISTAN = SQRT (DISTAN)

5.    ODD = 0.0
      EVEN = 0.0
      DO 50 K = 2, N, 2
      EVEN = EVEN + F (K)
50    ODD = ODD + F (K + 1)
      SUMM = F (1) + 4.* EVEN + 2.* ODD - F(N + 1)
      SIMPS = (B - A) * SUMM/(3. * N)

6.    ALFA (2) = B (1)/C(1)
      BETA (2) = F (1)/C (1)
      NM = N - 1
      DO 61 I = 2, NM
      DENOM = C (I) - A (I) * ALFA (I)
      ALFA (I + 1) = B (I)/DENOM
61    BETA (I + 1) = (A (I) * BETA (I) + F (I))/DENOM
      DENOM = C (N) - A (N) * ALFA (N)
      Y (N) = (A (N) * BETA (N) + F (N))/DENOM
      DO 62 IDASH = 1, NM
      I = N - IDASH
62    Y (I) = ALFA (I + 1) * Y (I + 1) + BETA (I + 1)

```

§ 11.

```

1.    DELTA (A, B, C, D) = A * D - B * C
      . . . . .

      DET = DELTA(A(1, 1), A(1, 2), A(2, 1), A(2, 2))
      DETX = DELTA(B(1), A(1, 2), B(2), A(2, 2))
      DETY = DELTA(A(1, 1), B(1), A(2, 1), B(2))
      X = DETX/DET
      Y = DETY/DET

2.    TERM(A, B, X) = A + B * X
      . . . . .
      B = A(5)
      DO 20 IDASH = 1, 4
      I = 5 - IDASH
20    B = TERM (A(I), B, X)
      P4X = B

3.    См. пример в п. 19. 4, стр. 159

4.    FUNCTION AVER(A, N)
      DIMENSION A(N)
      AVER = 0.0
      DO 10 K = 1, N
10    AVER = AVER + A(K)
      AVER = AVER/N
      RETURN
      END

```


Цена 57 коп.

367
K265

2